UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA
Depto de Eng. de Computação e Automação Industrial

LINHAS E SUPERFÍCIES ESCONDIDAS: TAXONOMIA E PROPOSTA DE IMPLEMENTAÇÃO VLSI DE UM ALGORITMO Z-BUFFER

6 de abril de 1990

Por: Eng. Luis Aldomiro Logatti
Orientador: Prof. Dr. Léo P. Magalhães

Tese de Mestrado apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas como parte dos requisitos para título de Mestre em Engenharia Elétrica.

Resumo

A determinação da visibilidade de objetos e' fundamental em síntese de imagens. No início dos anos 60 o primeiro algoritmo de eliminação de linhas foi desenvolvido e, desde la', vários algoritmos foram propostos. Este trabalho apresenta uma série de algoritmos de eliminação de linhas e superfícies escondidas, bem como duas classificações segundo os mais importantes artigos nesta área. Uma proposta de implementação do algoritmo Z-buffer em gate-array e' o tema principal da dissertação.

Agradecimentos

Gostaria de agradecer aos meus pais Francisco e Lucia Logatti por todo o incentivo que me deram durante toda a vida para eu que aprendesse cada vez mais.

Ao Ronaldo Kondo pela grande ajuda na implementação do circuito.

Ao Danilo Holanda pelo excelente trabalho com os desenhos.

À IBM pelo suporte recebido.

À Ivete pela paciência e pelo incentivo.

Conteúdo

Lista de Ilustrações

1.0 Introdução

O computador e´ uma máquina de processamento e armazenamento de informações; com ele e´ possível gerar, manipular e armazenar dados como nunca foi possível antes. Toda esta capacidade de processamento de informações nos permitiu a criação de novas aplicações tais como controle de tráfego aéreo, sistema bancário on-line, projeto auxiliado por computador, etc. No entanto, e´ desejável que esta informação seja apresentada numa forma que possa ser facilmente assimilada pelo ser humano .

Um gráfico pode substituir grandes quantidades de números ou ainda uma imagem pode apresentar mais informação que pilhas de relatórios; portanto, a capacidade de um computador manipular, criar e armazenar imagens facilita a comunicação homem-máquina, fazendo com que uma série de novas aplicações seja criada, quer no processo de gerência, como a apresentação de gráficos e diagramas; na produção científica como auxílio na criação de modelos; na educação com a criação de filmes educativos de fenômenos científicos, para acelerar o processo de aprendizado.

Desde a década de 70, a computação gráfica vem sendo largamente usada nos vários campos de atuação da vida moderna, desde os populares videogames até as ferramentas de auxílio ao projeto e ao diagnóstico passando por funções complexas de simuladores de vôo. Várias áreas do conhecimento humano foram auxiliadas pelo desenvolvimento da capacidade gráfica de computadores. Podemos citar, por exemplo, o projeto de circuitos integrados, que hoje e´ inconcebível sem as ferramentas de auxílio ao projeto. A indústria cinematográfica foi também beneficiada com a computação gráfica através de efeitos especiais. A série "Guerras nas Estrelas", com três filmes, está entre os dez maiores sucessos de bilheteria de todos os tempos. Grande parte destes filmes foi desenvolvida através do uso de software e hardware próprios para geração de imagens (a produtora dos filmes Lucas Films LTD. possui uma série de trabalhos publicados na área da computação gráfica).

A computação gráfica aplicada ao cinema talvez seja a parte mais fascinante para o grande público mas o seu uso nas ferramentas de auxílio ao projeto (mecânico, elétrico ou eletrônico) e ao diagnóstico tem feito com que ela evolua muito viabilizando assim novas aplicações.

A evolução da computação gráfica está intimamente ligada aos dispositivos de hardware tais como CPU´s e dispositivos periféricos, principalmente dispositivos de saída (displays). Em 1950, o primeiro monitor usado com o computador Whirlwind do MIT gerou imagens simples, usando Tubo de Raios Catódicos (TRC) igual ao empregado em aparelhos de TV. Muito do progresso em computação gráfica pode ser creditado aos avanços dos dispositivos de alta integração (VLSI) e as tecnologias de monitores. Graças aos microprocessadores e dispositivos de memória de baixo custo, um número crescente de funções esta´ sendo executado economicamente em menos espaço e com nível de sofisticação cada vez maior.

1.1 Síntese de Imagens

A relação entre entrada e saída em sistemas que criam, manipulam ou analisam imagens esta´ representada na Figura 1 na página 2:

E / S	IMAGEM	DESCRICAO
IMAGEM	MANIPULACAO IMAGENS	COMPUTACAO GRAFICA GERATRIZ
DESCRICAO	ANALISE IMAGENS	OUTROS

Figura 1. Relação entrada/saída de um sistema

Quando a entrada do sistema e' composta por imagens e sua saída também o e', dizemos que a função realizada pelo sistema e' a de processamento de imagens. Como exemplos destes sistemas podemos citar aqueles que são alimentados por câmeras de televisão, e que tem como resultado imagens compostas ou ainda efeitos visuais.

Ja' quando um sistema e' alimentado com imagens e tem como resultado uma descrição qualquer, dizemos que a tarefa que este sistema realiza e' a de análise de imagens ou de visão computacional. Podemos citar como aplicações destes sistemas a análise de fotos aéreas, tomografia computadorizada, análise de composições moleculares, etc.

O sistema que tem como entrada uma descrição de objetos e como saída imagens e' um sistema de síntese de imagens. Estas imagens são úteis em aplicações onde e' desejável visualizar um objeto sem precisar construí-lo fisicamente. Esta área e' chamada de computação gráfica geratriz ou simplesmente computação gráfica.

Um sistema de síntese de imagens e' constituído de várias partes, ou seja, módulos que têm como objetivo final gerar uma imagem a partir de uma descrição de objetos em um dispositivo de saída. A idéia básica e' simular dentro do ambiente computacional todos os fenômenos físicos de propagação, reflexão e difração da luz sobre objetos.

Baseado na simulação da propagação da luz o problema da síntese de imagens pode ser dividido em:

- modelamento geométrico da cena;
- determinação da visibilidade dos objetos;
- efetivação do rendering da cena.

O problema da determinação da visibilidade dos objetos é fundamental na geração de imagens, pois a apresentação destes objetos numa imagem resulta de dependencias entre os mesmos bem como do ponto de observação do observador, além da análise da distribuição de luz. Um objeto pode esconder outro ou ainda parte de si mesmo

dependendo do ponto de observação do observador. O problema de visibilidade dos objetos e' também chamado de problema da eliminação de linhas e superfícies escondidas.

O modelamento da iluminação no ambiente e' a parte do problema que estuda a distribuição de luz na cena, onde esta luz pode ser proveniente das fontes diretas ou ainda refletida por outros objetos em cena.

O processo de coloração leva em conta as propriedades de cada objeto para determinar a luz que reflete em cada objeto.

A Figura 2 nos apresenta resumidamente em forma de diagrama o que foi discutido anteriormente:

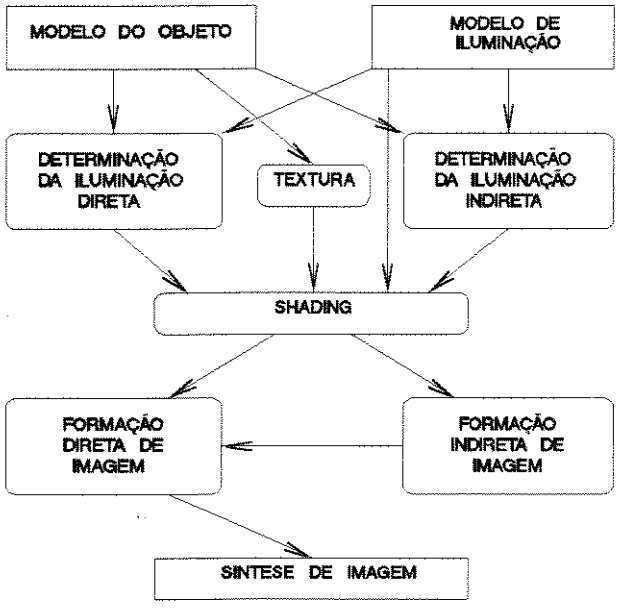


Figura 2. Modelo proposto por /Joy88/

1.2 Determinação da Visibilidade dos Objetos na Síntese de Imagens

A determinação da visibilidade dos objetos e' um problema fundamental em síntese de imagens. A determinação da visibilidade com um ponto de observação fixo num dado instante e' o mais comumente encontrado. Os algoritmos para solução deste problema de visibilidade formam a base para a solução de problemas mais complexos.

Na vida real, o material opaco de que são construídos os objetos obstruem os raios de luz das partes escondidas fazendo com que não as enxerguemos. Na síntese computadorizada de imagens esta eliminação automática não ocorre, ao contrário, muitas partes que não deveríamos ver são mostradas. Para eliminar estas partes e fornecer uma imagem mais realista são aplicados algoritmos de eliminação de linhas e superfícies escondidas.

No início dos anos 60, quando o primeiro algoritmo com esta finalidade surgiu, os displays eram dispositivos que desenhavam apenas vetores, chamados displays vetoriais e, portanto, o esforço era concentrado na remoção de linhas. Quando dispositivos raster tornaram-se disponíveis, a atenção foi voltada também para a remoção de superfícies.

O problema da determinação da visibilidade pode ser apresentado como:

DADO
  um conjunto de superfícies em 3D;
  um ponto de observação;
  um plano de imagens orientado;
  um campo de visão;
PARA CADA ponto no plano da imagem dentro do campo de visão
FAÇA
  determine qual ponto numa superfície reside mais próximo do
  ponto de observação ao longo de uma reta que passa pelo
  ponto de observação e um ponto no plano da imagem;
FIM;

É neste problema que concentraremos nosso trabalho.

1.3 Sistema Gráfico

Todas as técnicas usadas para a geração de imagens de cenas tridimensionais começam com um modelo. O modelo é necessário por dois propósitos: ele é usado pelo algoritmo juntamente com a informação da localização do observador para sintetizar a imagem da cena e para modificar e analisar os objetos em cena, atividades estas dos programas aplicativos.

A informação num modelo 3D pode ser dividida em geométrica, topológica e complementar. A primeira se refere à localização de um ponto ou ainda a dimensões de um objeto. A topológica se refere a como os pontos estão agregados para formar polígonos, por exemplo, como os polígonos formam os objetos e como os objetos formam cenas. A informação complementar refere-se a outro atributos dos objetos da cena tais como cor e intensidade.

O modelo geométrico de uma cena 3D deve ser representado em um sistema de coordenadas tridimensionais, onde a localização de um ponto é expressa por uma tríade (x,y,z), chamado sistema de coordenadas do "mundo".

A construção de um modelo de objetos é baseada num conjunto de primitivas, sendo o poliedro uma primitiva muito utilizada. Um poliedro arbitrário pode ser modelado pela definição de suas faces, onde cada face é um polígono plano que, por sua vez, pode ser modelado por uma lista de vértices ou por uma lista de bordas. Para geração de imagens de contorno apenas, as bordas do poliedro têm vital importância; para geração de imagens com linhas ou superfícies escondidas, no entanto, o aspecto mais importante do poliedro é sua face, pois é ela que esconde eventualmente outros objetos do observador e que deve ser sombreada apropriadamente.

A topologia e geometria de um modelo devem ser representadas num sistema computacional de uma maneira que permita o uso desta informação pelos algoritmos que geram imagens. A estrutura de dados depende do algoritmo escolhido, embora existam pontos comuns a todos eles.

Vejamos um exemplo da representação de um cubo da Figura 3 na página 5.

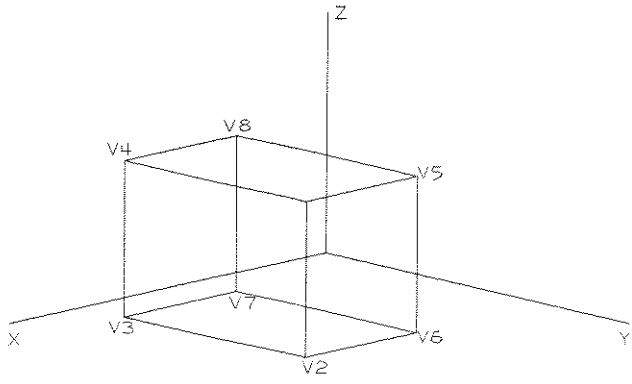


Figura 3. Exemplo de um cubo

Normalmente, um modelo terá mais informação associada do que a representada na Figura 4 na página 6. Geralmente inclui-se a estrutura hierárquica nesses modelos 3D, pois ela e' util no modelamento de qualquer repetição geométrica ou conectividade de um objeto. Na representação de um carro, por exemplo, podemos repetir o mesmo símbolo, as rodas, quatro vezes, apenas alterando a localização do mesmo.

Outro aspecto que diz respeito ao modelamento é a dificuldade de tratar problemas complexos. Para o uso dos algoritmos que descreveremos neste trabalho, o objeto deve ser aproximado por um conjunto de faces planas, caso o objeto já não tenha sua representação natural nesta forma. Após a definição do conjunto de faces que representa o objeto, torna-se necessário obter as coordenadas de seus vértices. Este processo pode ser feito através da digitalização, fazendo uso de digitalizadores, ou como resultado de algum cálculo computacional. A obtenção desses dados é vital para o bom funcionamento dos programas aplicativos. A dificuldade de construir modelos aumenta com a quantidade de informações topológicas requeridas pelos algoritmos.

GEOMETRIA

VERTICES
COORDENADAS X,Y,Z

V1 (1,1,1)
V2 (1,1,-1)
V3 (1,-1,-1)
V4 (1,-1,1)
V5 (-1,1,1)
V6 (-1,1,-1)
V7 (-1,-1,-1)
V8 (-1,-1,1)

EQUACOES PLANAS
AX+BY+CZ+D>O SIGNIFICA FORA

F1 [0,0,1,-1]
F2 [-1,0,0,-1]
F3 [0,0,-1,-1]
F4 [1,0,0,-1]
F5 [0,-1,0,-1]
F6 [0,1,0,-1]

TOPOLOGIA

F1 V1,V5,V8,V4
F2 V5,V6,V7,V8
F3 V6,V2,V3,V7
F4 V1,V4,V3,V2
F5 V8,V7,V3,V4
F6 V6,V5,V1,V2

BORDAS
PODEM SER DERIVADAS DAS FACES
MAS AS DUPLICATAS SAO REMOVIDAS

V1,V4 V7,V8
V4,V3 V8,V5
V3,V2 V5,V1
V2,V1 V8,V4
V5,V6 V6,V2
V6,V7 V7,V3

INFORMACAO AUXILIAR

CORES
COMPONENTES VERMELHOS,VERDES,AZUIS

F1 (0,4,0,0)
F2 (0,4,0,0)
... TODAS AS FACES SAO DA MESMA COR

Figura 4. Informações do cubo da Figura 3

A estrutura de dados contém a descrição da cena composta por objetos, sendo que cada um deles e' descrito por um conjunto de polígonos que, por sua vez, estão descritos pelas coordenadas de seus vértices expressas no sistema de coordenadas do "mundo", juntamente com suas informações complementares.

A primeira operação no processo de linhas e superfícies escondidas e' a conversão das coordenadas dos vértices do sistema "mundo" para as coordenadas no sistema de coordenadas de visualização, de acordo com a posição do ponto de observação e de sua direção. O polígono e' então recortado (clipping) segundo a pirâmide de visualização, eliminando-se as partes que estão fora do campo de visão. Após esta operação, o polígono pode ser projetado sobre o plano de projeção e expresso em coordenadas do dispositivo gráfico. Os cálculos de iluminação, tais como a determinação dos componentes da imagem que são dependentes da luz, através da imagem e distribuição de luz refletida , são executados em conjunto com a determinação da visibilidade das superfícies.

Transformações geométricas são usadas no modelamento para expressar a localização dos objetos em relação aos outros e transformações de perspectiva são usadas para projetar uma cena tridimensional para um plano bidimensional, que representa a superfície de visualização do dispositivo gráfico. Em aplicações onde o ponto de observação muda rapidamente ou ainda onde os objetos se movem em relação aos outros, estas transformações são executadas repetidamente.

1.4 Descrição Geral do Trabalho

Este trabalho está dividido em sete capítulos. O primeiro deles constitui esta introdução, onde foi colocado de maneira sucinta o problema que será discutido nos próximos capítulos. O segundo capítulo descreve os principais algoritmos de eliminação de linhas e superfícies escondidas, apresentando também uma comparação baseada em dois dos mais conhecidos artigos nesta área. No capítulo três, daremos uma introdução ao projeto de circuitos VLSI, mostrando as fases de projeto.

No capítulo quatro está descrita a proposta de implementação do algoritmo z-buffer em hardware. Os capítulos cinco, seis e sete encerram este trabalho contendo a proposta de implementação, conclusões e bibliografia respectivamente. O Anexo apresenta os resultados das simulações, alem de uma descrição do sistema de projetos EDS.

2.0 Taxonomia de Algoritmos para Eliminação de Linhas e Superfícies Escondidas

2.1 Introdução

Como foi visto no capítulo anterior, o problema de determinação da visibilidade de objetos num sistema de síntese de imagens e' fundamental. Vários autores apresentaram suas taxonomias a respeito do assunto. Neste trabalho iremos descrever apenas os algoritmos de maior importância, tanto historicamente como por sua aplicação prática, para depois apresentarmos a classificação de dois importantes trabalhos nesta área, /Suth74/ e /Joy89/.

2.2 Algoritmo de Roberts

Apresentamos este algoritmo por seu valor histórico. L. G .Roberts /Rob63/ criou a primeira solução ao problema de superfícies escondidas. Seu algoritmo testa cada aresta relevante para verificar se ela está obstruída pelo próprio objeto ou por um outro que compõe a cena. Este teste e' implementado escrevendo-se uma equação paramétrica para a linha de um ponto da aresta até o ponto de observação:

$$P=(1-\alpha)E1 + \alpha E2 + \beta(\ 0\ 0\ 1\ 0\ ) \text{ onde } 0\leq\alpha\leq 1 \text{ e } \beta\geq 0$$

Os dois primeiros termos são simplesmente a equação paramétrica de um ponto sobre aresta E1-E2. O terceiro termo e' um vetor apontando em direção ao ponto de observação $(0,0,\infty)$.

O ponto P reside dentro do objeto convexo, se P estiver dentro de todos os planos que o interferem, ou seja:

$P*Fij \leq 0$, onde Fij e' a equação do plano da i-ésima face do objeto j

Portanto se para um dado objeto j, os valores de α e β podem ser encontrados de forma a satisfazer a equação acima, o ponto na aresta correspondente a α está escondido pelo mesmo.

O algoritmo usa uma série de técnicas para resolver a equação acima, tais como testes MINIMAX e, caso estes não sejam suficientes, técnicas de programação linear.

As limitações deste algoritmo são as seguintes:

- a dificuldade de manipular objetos côncavos, pois a representação de objetos côncavos por convexos e' uma tarefa árdua;
- o tempo de computação cresce proporcionalmente com o quadrado do número de objetos em cena.

2.3 *Algoritmos de Appel (1967) e de Loutrel(1967)*

Estes algoritmos também são conhecidos como algoritmos de intersecção de arestas. Appel /App67/ define o termo "invisibilidade quantitativa" (IQ), assim como Loutrel /Lou67/ define o termo "ordem de invisibilidade" de um ponto como sendo o número de faces relevantes existentes entre o ponto e o ponto de observação.

Dois outros conceitos definidos por Appel são:

- arestas materiais ou relevantes: são aquelas que delimitam uma face potencialmente visível;
- arestas de contorno: são aquelas partilhadas por uma superfície invisível e uma potencialmente visível.

A solução do problema de linhas escondidas requer o cálculo da IQ para todos os pontos de cada aresta relevante. Primeiro são eliminadas todas as arestas que estão escondidas pelo objeto, através do uso de coerência de arestas. Num segundo passo, a visibilidade das arestas é calculada. A invisibilidade quantitativa e' calculada para o vértice inicial através da busca de todas as faces relevantes; posteriormente é computado quantas faces o escondem e a partir daí, a invisibilidade e' decrementada ou incrementada de 1 para os demais pontos, dependendo se a aresta passa à frente ou por trás de uma aresta de contorno. Desta forma evita-se o cálculo exaustivo de busca do vértice inicial, que para os demais pontos e' realizado apenas uma vez e, subsequentemente, são realizados cálculos incrementais a partir do mesmo.

O método de Loutrel e' muito similar ao de Appel. Loutrel considera a intersecção de arestas pela projeção de arestas no plano da imagem e calcula a intersecção num espaço bidimensional. Se uma intersecção e'encontrada, as profundidades das duas arestas são comparadas neste ponto para decidir qual esconde a outra.

Cada um destes algoritmos localiza todas as intersecções ao longo de uma aresta relevante e então um ordenamento é executado e sua invisibilidade e' determinada. Muitas vezes o ordenamento pode ser evitado detectando-se que a invisibilidade do vértice inicial e' tão alta que não ha' intersecções suficientes para torna'-lo visível .

2.4 *Algoritmo de Warnock (1969)*

O algoritmo de Warnock /War69/ assume que áreas de amostragem, chamadas de janelas (windows), podem ser consideradas homogêneas e podem ser mostradas após um simples cálculo de visibilidade.

Esta hipótese e' considerada correta se:

- não existem faces dentro da janela de amostragem ou
- uma face cobre completamente as outras e esta' mais próxima do ponto de observação do que todas as outras faces que estejam dentro da janela.

Se estas condições não podem ser provadas ou aparentam ser de difícil prova, a janela de amostragem e'subdividida em quatro janelas menores e a hipótese e' reexaminada repetindo-se a operação. Quando o tamanho da janela atinge o tamanho de um elemento de rastreio (raster) a recursão e' terminada.

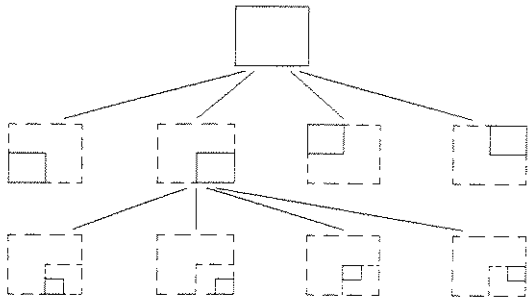


Figura 5. Subdivisão recursiva de janelas do algoritmo de Warnock

Um conjunto de faces e' comparada com a janela para verificar se este:

- esta' completamente fora da janela, portanto faces e janela são disjuntas;
- esta' contido parcialmente na janela, portanto faces e janela são interceptoras;
- contém completamente a janela, portanto faces circundam a janela.

As faces são divididas em dois grupos:

- aquelas que são disjuntas;
- aquelas que são relevantes (interceptoras e circundantes).

A cada recursão as faces são reclassificadas e as faces relevantes são analisadas e repassadas ou não para as demais subdivisões.

A análise das relações de polígonos com a janela aumenta a velocidade de análise das sub-janelas. Por exemplo, se um polígono e' disjunto de uma janela, ele sera' disjunto de todas as sub-janelas. Portanto, se a recursão vai se aprofundando as janelas vão ficando cada vez menores e, consequentemente, a lista de polígonos relevantes vai diminuindo.

Quando numa janela for encontrado um polígono circundante e um interceptor, o algoritmo deve decidir qual caso de homogeneidade existe. Caso não exista nenhum polígono, a janela está vazia e deve ser preenchida com a cor de fundo. Caso um polígono circundante seja encontrado, o algoritmo busca a face crítica, ou seja, aquela que está mais próxima do ponto de observação. Caso haja polígono circundante que penetre em outro, a janela não e' homogênea e nova recursão é necessária.

Existem variações do algoritmo de Warnock. Um método e' criar sub-janelas não-quadradas, que sigam o formato do polígonos mais externos. Podemos ainda sudividir as janelas em pontos específicos, tais como os vértices, ao invés do centro da janela.

2.5 *Algoritmos Scan-line*

Dentro desta categoria podemos incluir os algoritmos de G. S. Watkins (1970), W. J. Booknight (1969) e C. Wylie, G. W. Romney, D.C. Evans, A. Erdahl (1967).

Estes algoritmos também são chamados de algoritmos de coerência de segmento (span). Eles usam uma forma unidimensional de coerência de área. Pequenos spans, ou sequências de pixels numa mesma linha de varredura residirão num mesmo polígono. O algoritmo busca um span no qual ele precise fazer apenas algumas comparações de profundidade para identificar se o polígono e' visível ou não.

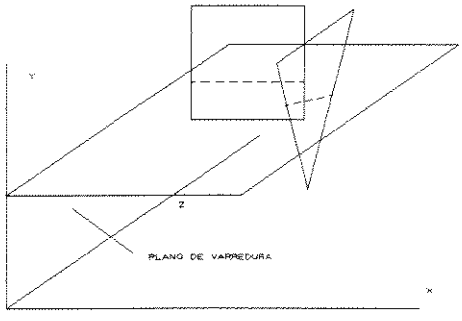


Figura 6. plano de varredura

Cálculos no plano xz são usados para determinar as relações entre partes de todos os polígonos interceptados pela linha de varredura e decidir quais partes são visíveis e em quais spans.

As coordenadas x e z dos pontos limites de um segmento podem ser obtidas de uma maneira incremental, pois elas são função linear de y, isto e', a equação da aresta do polígono pode ser escrita como:

$$x = \alpha * y + \beta \quad e$$

$$z = \gamma * y + \delta$$

Os spans de uma linha de varredura são identificados pelo ordenamento em x de todas as extremidades dos segmentos. São então examinadas as regiões delimitadas por estes limites, como podemos ver na figura Figura 7 na página 12:

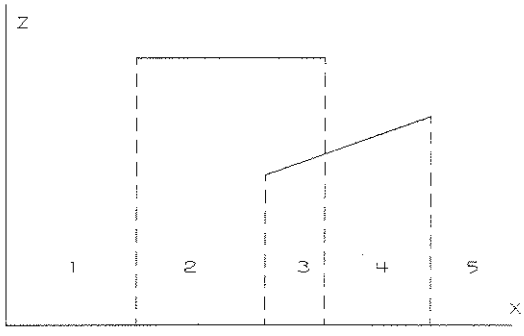


Figura 7. Segmentos dos polígonos no plano x e z

Spans, que na figura são os segmentos marcados 1, 2, 3, 4 e 5, podem ser classificados em uma destas 3 classes:

- nenhum dos segmentos aparece dentro do span, a intensidade do fundo e' mostrada (1 e 5);
- apenas um segmento está dentro do span, obviamente este segmento e' visível (2 e 4);
- vários segmentos aparecem; o segmento mais próximo do ponto de observação (z min) e' encontrado pela comparação das profundidades de todos os segmentos de uma coordenada x, o polígono identificado como o mais próximo e' mostrado (3).

O algoritmo desenvolvido por Romney et ali /Romn70/ foi o primeiro a usar estas características. O ordenamento da coordenada y (varredura de cima para baixo e esquerda para a direita) usa "bucket sort" (Figura 8 na página 13). Uma dada scan-line, com uma coordenada y, e' usada para atualizar a tabela de ocupação de y na qual estão armazenadas todas as faces que a interceptam.

Estas interseções das faces com a scan-line são ordenadas dentro do bucket de x, sendo mantida uma lista das faces potencialmente visíveis. Cada vez que uma face entra ou deixa a tabela de ocupação x, o algoritmo recalcula as profundidades de todas as faces na tabela para encontrar qual e' a mais próxima. Esta decisão persiste até a próxima mudança.

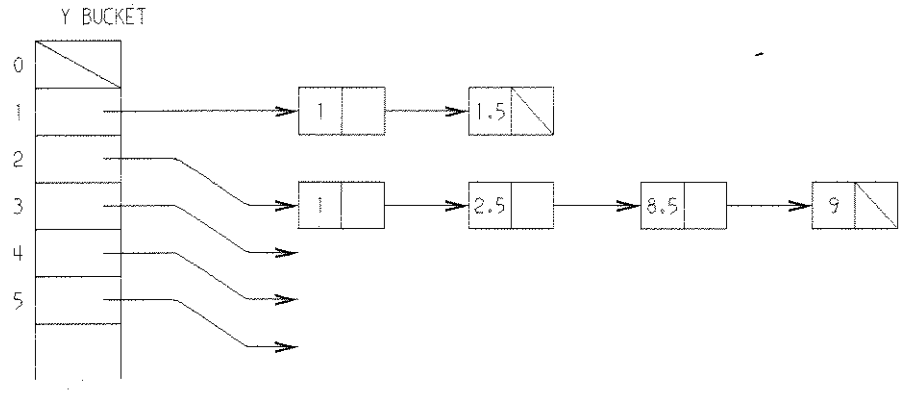


Figura 8. **Bucket sort**

2.6 Algoritmo do Pintor (Painter's Algorithm)

O algoritmo do pintor tem este nome devido a analogia que se faz a um pintor que, usando tinta opaca, pinta os objetos mais próximos do observador sobre os mais distantes.

2.6.1 Algoritmo de R. E. Newell, R. G. Newell e T. L. Sancha (1972)

A idéia geral deste algoritmo e' traçar as imagens de sucessivas faces com prioridades num buffer de imagem. Faces com prioridades maiores irão se sobrepor a faces com prioridades menores. Portanto, teremos a visão correta com superfícies apropriadamente escondidas após o processamento de toda a lista.

O cálculo de prioridade e' efetuado pelo ordenamento de todos os polígonos em cena baseado na coordenada z (profundidade) do ponto mais distante do ponto de observação de cada polígono. Se após o ordenamento, não existirem polígonos adjacentes na lista que se sobreponham, a lista está na ordem correta.

A sobreposição de dois polígonos pode ser determinada por testes, tais como o MINIMAX. Este teste consiste em verificar se a coordenada x mínima de um polígono e' maior que a coordenada x máxima de outro. Se isto ocorre, há a possibilidade de sobreposição. O mesmo argumento e' válido para as coordenadas y. Este tipo de teste agiliza o processamento evitando-se cálculos mais detalhados, que deverão ser executados caso este venha a falhar.

Se polígonos da lista de prioridade se sobrepõem em profundidade, cálculos mais detalhados são necessários para determinar a prioridade correta.

Considere P o último polígono da lista. Se P não se sobrepõe em profundidade ao seu predecessor na lista, Q, então P não possui nenhuma sobreposição com nenhum dos polígonos da lista e e' o polígono de menor prioridade.

Suponha agora que P se sobreponha em profundidade a um conjunto de polígonos N, que o precedem na lista. Para determinar este conjunto, e' necessário percorrer toda a lista de volta até que não se ache mais nenhum polígono que seja superposto por P.

P somente irá esconder Q da lista de polígonos se um dos seguintes testes for verdadeiro:

- Teste de profundidade MINIMAX indica que P e Q não se sobrepõem em z (profundidade);
- Teste MINIMAX de xy indica que P e Q não se sobrepõem em x ou y;
- Todos os vértices de P são mais distantes do ponto de observação que o plano de Q. Este teste e' implementado pela substituição das coordenadas xy dos vértices de P na equação do plano Q e resolvendo a mesma para a profundidade de Q;
- Todos os vértices de Q são mais próximos do ponto de observação que o plano P;
- Um teste completo de sobreposição indica que P e Q não se sobrepõem.

Estes testes são aplicados na ordem descrita acima, porque eles se tornam cada vez mais difíceis de serem executados do que seus antecessores.

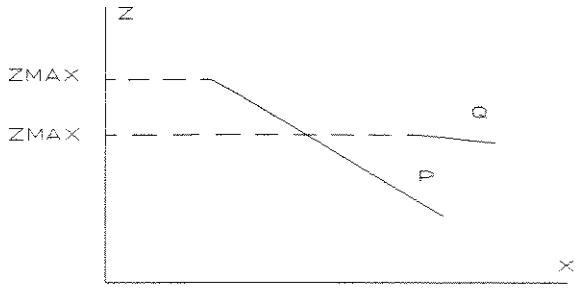


Figura 9. A prioridade de Q e' maior que P numa primeira análise

A relação "P esconde Q" (teste descrito anteriormente) não e' suficiente para o ordenamento dos polígonos. Quando um polígono e' movido na lista, ele e' marcado para evitar uma nova tentativa de movê-lo. Este polígono pode então ser dividido em duas partes na tentativa de resolver o problema, criando-se assim polígonos com prioridades diferentes. O polígono original e' descartado e suas partes são acrescentadas em ordem na lista e o algoritmo prossegue como antes, tratando os dois novos polígonos.

Temos como exemplo a Figura 10 na página 16, onde o polígono Q foi dividido em Q1 e Q2.

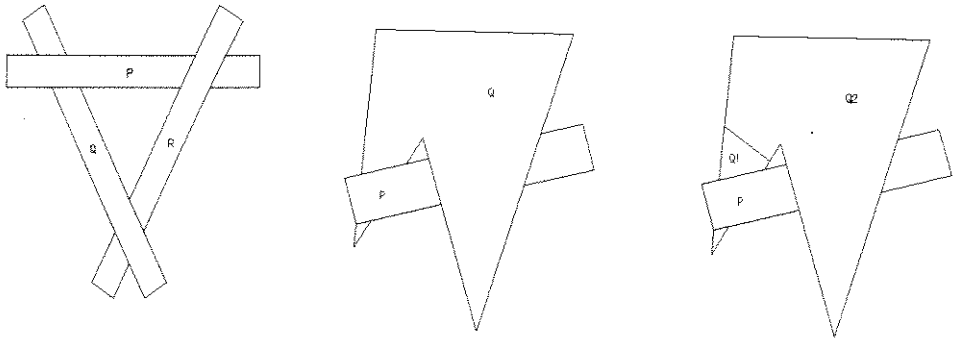


Figura 10. **Casos de prioridades: overlap cíclico, sobreposição e divisão de polígonos**

2.6.2 Algoritmo de R. A. Schumacker, B. Brand, M. Gilliland & W. Sharp (1969)

A diferença básica entre os algoritmos de Schumacker e de Newell está no cálculo da lista de prioridades. Enquanto o primeiro realiza a maioria dos cálculos de prioridade "off-line", o segundo o faz antes de processar cada frame. Embora o investimento em calcular a lista de prioridades a partir da descrição do ambiente seja alto, virtualmente a mesma lista pode ser usada em vários frames.

Schumacker observou que o cálculo da prioridade não precisa comparar cada face com todas as outras faces para se determinar a prioridade de cada. A imagem pode ser dividida em clusters (aglomerações) e dentro de cada cluster cada face pode ser comparada com todas as outras faces do mesmo cluster para se determinar a prioridade de face. Se a imagem possuir apenas um cluster, a lista de prioridades está completa;

caso contrário, o algoritmo calcula a prioridade relativa dos clusters, ou seja, a prioridade de cada cluster.

Se um objeto A está mais próximo do ponto de observação de que um objeto B, todas as faces de A têm prioridade sobre todas as faces de B, o que no entanto não e' válido quando A e B se interpenetram ou não são linearmente separáveis. A restrição acima e' aplicável ao uso do algoritmo de Schumacker: assim somente ambientes com faces convexas e clusters linearmente separáveis são permitidos.

A prioridade das faces dentro de um cluster pode ser determinada independentemente do ponto de observação, portanto uma vez determinada não se altera com a mudança do mesmo (Figura 11). Esta característica do algoritmo faz com que ele seja usado em simuladores de vôo, onde a imagem raramente muda, embora o ponto de observação o faça frequentemente.

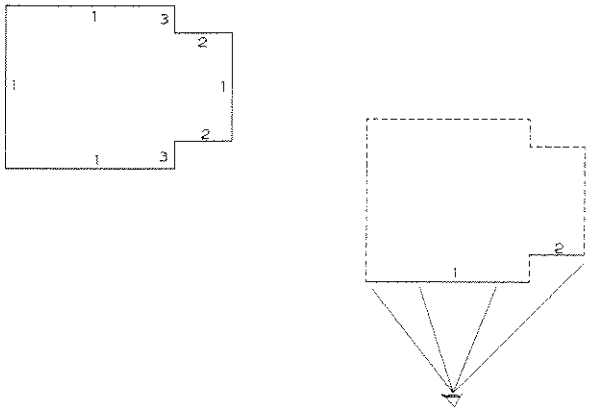


Figura 11. A localização do pto. de observação determina as faces de trás

Para se calcular a prioridade das faces e' necessário saber se a face A pode, de qualquer ponto de observação, esconder a face B; caso isso seja possível A tem prioridade sobre B. Se existirem circuitos no grafo, ou seja, face A tem prioridade sobre B e vice-versa, torna-se necessário dividir o cluster em pedaços menores para se fazer a determinação de prioridades.

Tomando a Figura 12 na página 18, vejamos como o cálculo das prioridades e' executado: considere A, B, C e D como pontos de observação. Considere que o ponto de observação esteja em C. Neste caso a prioridade de 3 e' maior que 2 que e' maior que 1. Este cálculo pode ser formalizado decidindo-se onde o ponto de observação reside em relação aos planos que separam os clusters (α,β). Caso o ponto de observação resida em C, por exemplo, as prioridades serão 3, 2, 1: pois 3 reside no mesmo plano que C, e em relação a β, 2 está no semiplano positivo, e finalmente 1 que não se encontra em nenhum semiplano de C. Podemos tomar o ponto A como o ponto de observação: neste caso teremos a prioridade 1, 2, 3 pois A esta' no mesmo plano que 1, no mesmo semiplano positivo de α e o cluster 3 aparece como último na lista de prioridades.

A árvore (b) da Figura 12 na página 18 mostra as quatro possíveis ordens de prioridades dos quatro clusters em relação ao ponto de observação.

Este conceito pode ser estendido a qualquer coleção de clusters, desde que planos de separação possam ser determinados. Alternativamente, os planos podem ser determinados dinamicamente com o movimento dos clusters, desde que a propriedade acima não seja violada.

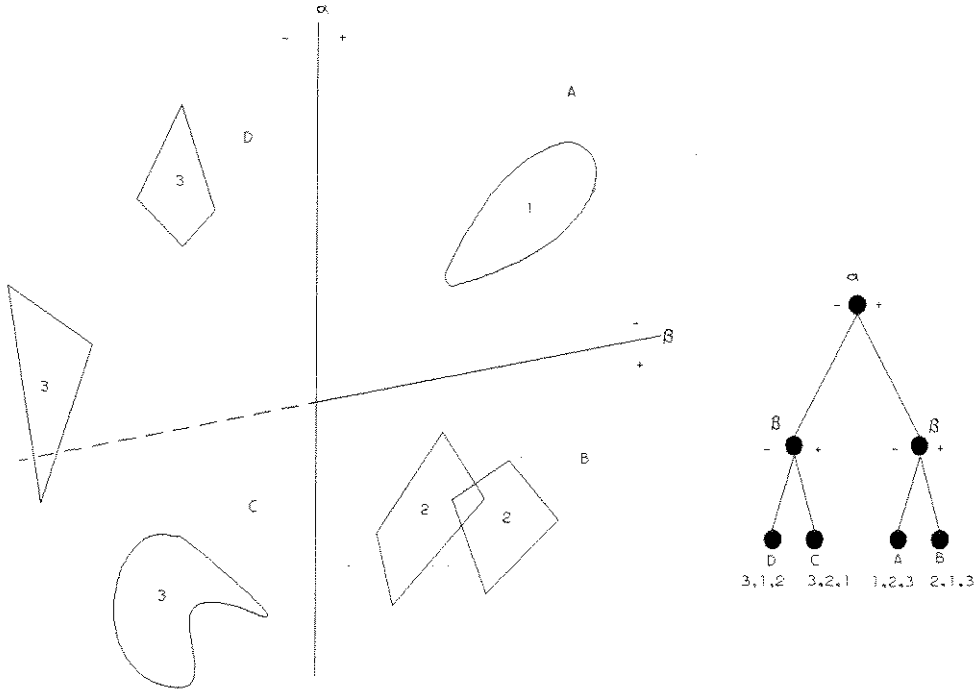


Figura 12. Prioridade de clusters

2.7 Algoritmo Ray-tracing

O método usado pelo algoritmo ray-tracing tem suas raízes no trabalho de Appel, apresentado anteriormente. O algoritmo emprega o princípio da óptica geométrica. Para cada pixel, um raio de largura infinitesimal e' traçado desde o ponto de observação ate' o espaço objeto, o raio e' testado em cada objeto na estrutura de dados e o objeto cuja intersecção e' a mais próxima do ponto de observação e' usado para o cálculo da intensidade do pixel na tela.

```
BEGIN
  FOR EACH pixel P
  DO
   forme um raio R no espaço objeto através da posição
        da câmera  e o pixel
   intensidade = traço(R)
   use intensidade para colorir o pixel P
  END
END;


PROCEDURE traço(raio)

   BEGIN
     FOR EACH entidade E em cena
     DO
      calcule as intesecções de E e o raio (se alguma)
      IF o raio não cruza nenhuma entidade THEN
        RETURN (intensidade de fundo)
      ELSE
        ache a entidade E com a intesecção mais próxima
        calcule a intensidade I no ponto de intersecção
        RETURN(modelo de iluminação(I,traço(raio de
        reflexão), traço (raio de refração)))
      END
     END
   END;
```

O algoritmo apresentado acima leva em consideração efeitos de sombreamento e transparências dos objetos da cena. Por estas considerações o algoritmo e' recursivo como podemos observar no comando RETURN. No entanto, quando apenas estivermos interessados na resolução do problema de linhas e superfícies escondidas devemos simplificar o algoritmo com a retirada da recursividade.

2.8 Algoritmo Z-Buffer

O desenvolvimento deste algoritmo e' usualmente atribuído a Catmull /Catm74/, sendo classificado por muitos autores, como o mais simples entre os algoritmos para eliminação de linhas e superfícies escondidas.

Para cada pixel do display e' mantido um registro da profundidade da primitiva em cena que e' a mais próxima do observador, mais um registro associado de intensidade do objeto. Quando um novo polígono está para ser processado, o valor z (profundidade) e a intensidade são calculados para cada pixel residente na periferia do polígono. Se o valor z do novo polígono indica que ele está mais próximo do observador que o valor indicado no z-buffer, este novo valor e' armazenado bem como sua intensidade associada. Após o processamento de todos os polígonos, o resultado do buffer dos registros das intensidades e' mostrado.

O algoritmo e' apresentado a seguir:

```
GIVEN
   Lista de polígonos { P1,P2,.....Pn}
   um array de z-buffer(x,y) inicializado em +∞
   um array de intensidade (x,y)
BEGIN
    FOR EACH polígono P na lista de polígono
    DO
```

```
        FOR EACH pixel(x,y) que intercepta P
        DO
         calcule o valor z de P em (x,y)
         IF z-depth < z-buffer(x,y) THEN
           z-buffer (x,y) = z-depth
         DISPLAY o array de intensidade
        END
      END
  END
```

A simplicidade do algoritmo z-buffer faz com que ele seja apropriado para implementação em hardware. Este fato motivou o presente trabalho e a ele voltaremos nos capítulos 4 e 5.

2.9 Taxonomia dos Algoritmos segundo Sutherland et alli

/Suth74/ dividiu os algoritmos para eliminação de linhas e superfícies em três grandes grupos:

-algoritmos espaço-objeto;
-algoritmos espaço-imagem;
-algoritmos de prioridades de listas.

Algoritmos espaço-objeto são aqueles que realizam seus cálculos no espaço-objeto, ou seja os cálculos são executados usando uma precisão arbitrária, geralmente aquela disponível no computador que está executando o algoritmo. O objetivo destes cálculos e´ chegar ate´ a precisão que possa ser executada para que o objeto seja representado da melhor maneira possível . Como vantagem deste fato, podemos aumentar a imagem várias vezes sem perda da precisão, desde que a precisão do computador em uso seja maior que a do display (dispositivo gráfico).

Os algoritmos espaço-objeto concentram-se nas relações geométricas entre os vários objetos em cena, de maneira a determinar que partes do objeto são visíveis ou não. O tempo de computação deste tipo de algoritmo aumenta na medida em que cresce o número de objetos em cena.

Os cálculos dos algoritmos no espaço-imagem são executados com uma precisão menor que os do espaço-objeto. São limitados à precisão do dispositivo gráfico que sera´ utilizado. Em última análise, eles apresentam uma imagem da solução e seu tempo de computação tende a aumentar conforme aumenta a complexidade das partes visíveis da imagem.

Os algoritmos de prioridades de listas usam uma combinação de ambos descritos anteriormente e serão melhor explicados no item 2.9.3.

Esta classificação e´ apresentada na Figura 13 na página 21.

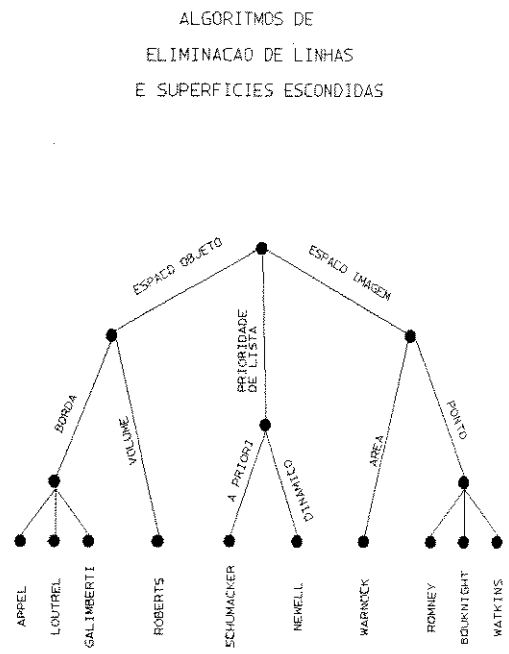


Figura 13. Taxonomia segundo /Suth74/

2.9.1 Algoritmos Espaço-Objeto

Entre os algoritmos espaço-objeto, podemos identificar uma outra divisão: os algoritmos de aresta e os de volume. Embora todos estes algoritmos testem arestas relevantes para determinar que partes destas arestas são visíveis, o critério da invisibilidade e' um pouco diferente. O algoritmo leva em conta a coerência espacial dos objetos, testando arestas contra objetos.

Os algoritmos de Appel e de Loutrel, entretanto, testam arestas contra arestas. Eles observam que a visibilidade de uma aresta e' coerente, particularmente nos vértices onde termina a mesma. Se a visibilidade de uma aresta e' calculada, ela pode ser usada para evitar cálculos nas outras arestas que partilham um vértice com a primeira. Desta maneira, a maioria das decisões de visibilidade tornam-se cálculos incrementais.

Os algoritmos espaço-objeto foram analisados em maior detalhe nos itens 2.2 e 2.3 deste capítulo. A bibliografia no capítulo 7 fornece os artigos principais dos mesmos.

2.9.2 Algoritmos Espaço-Imagem

Este classe de algoritmos pode ser dividida em duas categorias distintas:

- aqueles que amostram áreas da superfície de visualização (Warnock);
- aqueles que amostram pontos infinitesimais (algoritmos scan-line).

Chamaremos estes dois métodos de amostragem de área e amostragem de pontos, respectivamente. O objetivo do primeiro é calcular a intensidade apropriada de cada área da superfície de visualização. Se a superfície de visualização for homogênea, por

exemplo, este método exige apenas um cálculo, isto é, ele usa o princípio da coerência de área.

Já os algoritmos que utilizam a amostragem de ponto (scan-line), são projetados para resolver o problema de linhas e superficies escondidas de maneira a facilitar seu emprego por dispositivos gráficos tipo "raster". Estes algoritmos calculam a intersecção de um plano de uma linha de varredura com a face do objeto; estes segmentos de linha destas intersecções são chamados de spans.

A criação de segmentos simplifica o problema para o espaço bi-dimensional, pois os segmentos são medidos pelas coordenadas x e z. Esta redução de 3D para 2D faz com que sejam utilizados vários cálculos comuns tais como teste de segmentos para sobreposição ou profundidade.

Os algoritmos espaço-imagem foram analisados em maior detalhe nos itens 2.4 e 2.5 deste capítulo. A bibliografia no capítulo 7 fornece os artigos principais dos mesmos.

2.9.3 Algoritmos de Lista de Prioridades

Da mesma forma que os algoritmos que trabalham no espaço-imagem, estes algoritmos são utilizados para criar imagem para dispositivos gráficos com uma dada resolução, geralmente um dispositivo raster. No entanto, diferentemente dos algoritmos scan-line e Warnock, estes algoritmos primeiramente calculam a relação de profundidade e posteriormente executam os cálculos xy.

Estes algoritmos foram projetados visando aplicações em tempo real e onde o realismo da imagem e' um fator importante. Os grupos que trabalhavam no desenvolvimento destes algoritmos tinham em mente sua aplicação para sistemas de simulação de vôos para a NASA.

Como foi dito anteriormente, a distinção entre algoritmos de lista de prioridades e aqueles que trabalham no espaço-imagem reside na forma como a visibilidade final e' calculada. Como já vimos nos algoritmos que trabalham no espaço-imagem, o teste de visibilidade e' efetuado apenas no final, portanto estes algoritmos se utilizam da separação lateral da imagem para reduzir este tipo de cálculo.

Os algoritmos de Newell e de Schumacker, por outro lado, utilizam a idéia de rearranjar todos os polígonos em cena numa ordem de prioridade, baseada na profundidade deles, antes mesmo de gerar a figura no espaço-imagem. Polígonos mais próximos do ponto de observação têm prioridade maior que aqueles mais distantes.

Podemos então detectar características de ambos os tipos de algoritmos espaço-objeto e imagem nos algoritmos de lista de prioridades. Como os que trabalham no espaço-objeto, temos os cálculos de profundidade que são executados com alta precisão. E como os do espaço-imagem, temos que sua saída está diretamente ligada a resolução do dispositivo gráfico usado.

Os algoritmos espaço-imagem foram analisados em maior detalhe no item 2.6 deste capítulo. A bibliografia no capítulo 7 fornece os artigos principais dos mesmos.

2.10 Taxonomia dos Algoritmos segundo Joy et Allii

/Joy88/ dividiu os algoritmos nas seguintes categorias:

- algoritmos contínuos;
- algoritmos de amostragem de pontos.

Os algoritmos contínuos calculam a visibilidade dos objetos da cena através de áreas contínuas que cobrem toda a imagem. Estas áreas são utilizadas para varrer toda a cena, determinando assim quais regiões da imagem são visíveis. Se os objetos forem modelados por polígonos, estas regiões podem ser especificadas ate' a precisão do ponto flutuante do computador. Estes algoritmos se utilizam da coerência de área, como já foi explicado no algoritmo de Warnock: se um polígono e' visível num pixel geralmente ele o sera' no pixel adjacente.

Os algoritmos de amostragem de pontos formam uma solução aproximada para o problema de eliminação de superfícies escondidas. Existe uma aproximação sobre a visibilidade do objetos entre dois pontos de amostragem. São os seguintes os algoritmos de amostragem de pontos:

- z-buffer
- ray-tracing
- algoritmos do pintor

- algoritmo scan-line

Os algoritmos de amostragem de pontos foram analisados em maiores detalhes nos itens 2.6, 2.7 e 2.8 deste capítulo. A bibliografia do capítulo 6 fornece os artigos principais dos mesmos.

3.0 Introdução ao Projeto VLSI

3.1 Introdução

Neste capítulo veremos a evolução da indústria de semicondutores nestes últimos 40 anos, apresentaremos os diferentes estilos de projeto na área de circuitos integrados e também o fluxo de projeto para que possamos entender melhor a proposta de implementação apresentada nos capítulos 4 e 5.

3.2 A evolução dos semicondutores

A invenção do transistor em 1948 e a sua exploração comercial nos meados da década de 50 em rádios portáteis e computadores fez com que o desenvolvimento da indústria de eletrônica mudasse muito. A tendência do desenvolvimento da indústria, que parecia linear com o tempo, passou a ser exponencial.

Da fabricação de transistores discretos até a invenção dos primeiros circuitos integrados, em 1958, foi dado um grande passo. E, apenas no início da década de 1960, o uso de transistores conectados numa uma única peça de silício tornou-se comercial com os circuitos SSI (small scale integration), como eram chamados os primeiros circuitos integrados de baixa integração.

Não foi antes da metade da década de 60 que o processo de fabricação passou a ser entendido de maneira que a estabilidade e reproducibilidade das operações permitissem a construção de circuitos analógicos. Já no final da década de 60, a complexidade das funções digitais havia alcançado o patamar de MSI (medium scale integration), com circuitos com mais de 100 dispositivos, sempre utilizando o dispositivo básico, ou seja, o transistor bipolar que fora desenvolvido anos antes.

Foi quando um novo tipo de transistor, que iria mudar completamente a densidade dos circuitos integrados, surgiu. O MOS transistor (metal-oxide-semiconductor), que empregava um novo tipo de transporte de cargas elétricas, possuia características de alta impedância, baixa dissipação e alta compactação. O transistor unipolar, como era chamado o transistor MOS, foi na verdade o primeiro tipo de dispositivo a ser proposto anos antes mesmo da descoberta do efeito transistor pelos cientistas do Bell Labs. Um dos integrantes desta equipe estudava este dispositivo antes de se juntar ao grupo. No entanto, sua confecção era impossível com os recursos tecnólogicos existentes na época.

O aumento da complexidade dos circuitos integrados era buscado tanto utilizando-se dispositivos bipolares como unipolares, pois as pesquisas eram realizadas paralelamente. Os dispositivos bipolares inicialmente levaram grande vantagem em termo de velocidade, mas sua compactação era extremamente baixa: os dispositivos unipolares (MOS) apresentavam alta compactação, no entanto possuiam a característica de velocidade de chaveamento muito menor que seus similares bipolar. No entanto os dispositivos MOS nos permitiram atingir patamares de complexidade da ordem de milhões de dispositivos numa mesma pastilha e recentes pesquisas nos levam a crer que a diferença de velocidade de chaveamento em relação aos bipolares esta' diminuindo, ou pelo menos já atingiu um patamar muito menor que o encontrado anos atrás.

Com o desenvolvimento desta tecnologia e do processo de fabricação, em 1970 tivemos o lançamento das memórias RAM dinâmicas semicondutoras (DRAM), que ameaçou a posição dominante até então das memórias de núcleo de ferrite como elemento de memória principal de computadores. Com frequentes avanços na tecnologia, estas memórias aumentavam , a cada ano, sua capacidade e diminuíam seu custo por bit. Em menos de quinze anos, as primeiras memórias de 256k já eram fornecidas em grandes quantidades. Em fevereiro de 1986, durante a Conferência Internacional de Circuitos de Estado Sólido promovido pelo IEEE, a IBM anunciou o lançamento das memórias de 1 Megabit, com mais de um milhão de transistores, para no ano seguinte, na mesma conferência, anunciar em caráter experimental as memórias de 4 Megabits.

Estes aumentos constantes na complexidade destes dispositivos se devem basicamente às melhorias introduzidas no controle de processo de fabricação, permitindo assim que o yield, ou seja a quantidade de chips bons no final de cada processo, aumentasse muito. Esta evolução das memórias reduziu a dimensão mínima dos dispositivos de 10 microns em 1970 para até 0.1 microns nos dias de hoje, como já podemos encontrar em linhas experimentais dos laboratórios de desenvolvimento.

As memórias desempenham um papel muito importante no desenvolvimento de tecnologias pois apresentam uma demanda muito alta, sendo que este mercado pode ser considerado quase insaciável. Este altos volumes garantem o retorno do investimento aplicado em pesquisa e desenvolvimento. A tecnologia desenvolvida para memórias e' posteriormente empregada na implementação de dispositivos de lógica.

A evolução da tecnologia nesta área tem surpreendido até especialistas do setor. Pela lei de Gordon Moore, as memórias de 4M eram esperadas apenas em 1990. Gordon Moore, presidente da Intel, reunido com mais de 800 participantes durante um colóquio no Vale do Silício em 1977, apresentou a curva da Figura 14. Como podemos observar, esta curva, também chamada de primeira lei de Gordon Moore, previa que os circuitos com mais de 4 milhões de componentes funcionais (gates para lógica e bits para memória) so' apareceriam em 1990, no entanto foram ja' apresentadas em 1986.

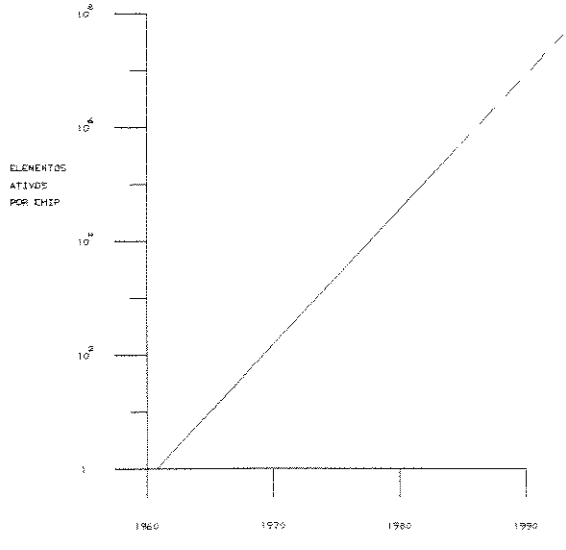


Figura 14. Representação gráfica da lei de Gordon Moore

Na Figura 15 na página 27 podemos observar a evolução das famílias dos componentes. No eixo y está representado o número de elementos ativos, que pode ser considerado uma porta lógica NAND para os elementos de lógica ou ainda um bit de memória.

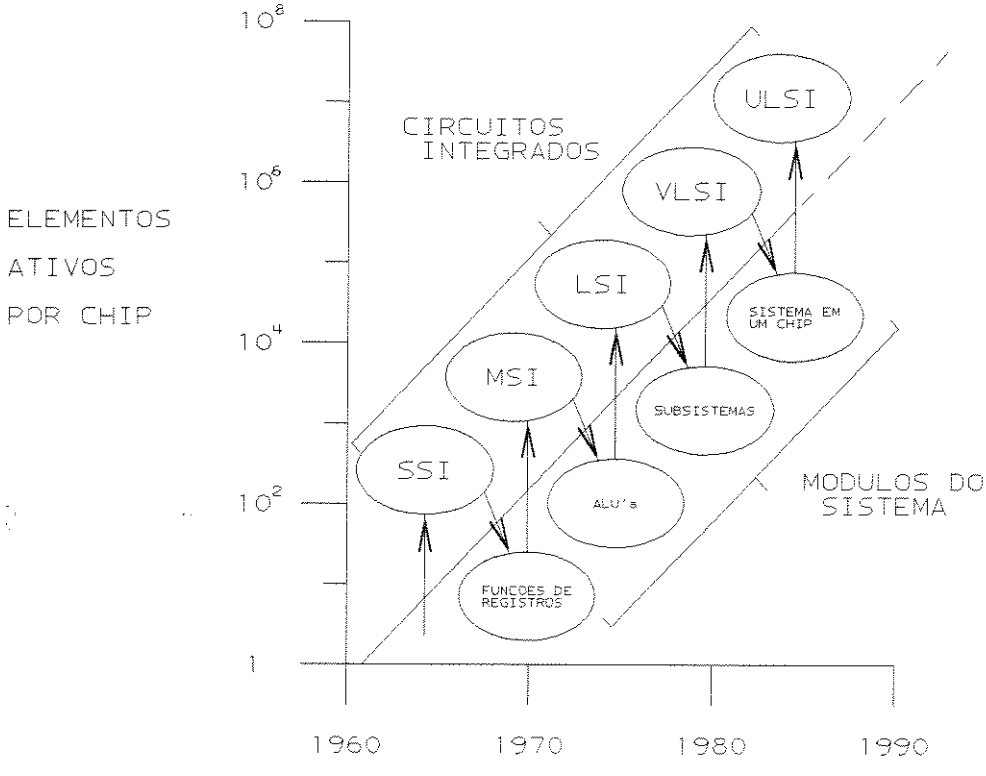


Figura 15. Evolução das famílias dos componentes

Como já foi dito anteriormente, os primeiros circuitos integrados possuíam baixa integração (SSI), geralmente com menos de 10 componentes por pastilha. Depois tivemos os circuitos que continham até cerca de 100 componentes por pastilha, ou seja, os circuitos de média integração, como os chamamos hoje (MSI). Com a rápida evolução da tecnologia de fabricação destes dispositivos passamos a ter até 1.000 componentes por pastilha (LSI) e hoje os circuitos possuem milhares de componentes ativos, chamados de circuitos de alta integração ou VLSI (very large scale integration). O termo ULSI, referente a circuitos de altíssima integração, apesar de corresponder a uma nova fase da tecnologia não é ainda amplamente usado. Podemos observar também na Figura 15, os módulos dos sistemas que passaram a ser totalmente integrados conforme a complexidade dos circuitos integrados ia permitindo. Hoje sistemas inteiros são integrados numa única pastilha. O estado da arte nesta área de tecnologia nos apresenta as memórias de 16 Megabits já comerciais para o próximo

ano e para a área de lógica temos processadores de mais de 20 MIPS integrados numa única pastilha.

3.3 *Estilos de Projeto VLSI*

Milhões de circuitos integrados são fabricados para consumo e a maioria e' parte de uma família que já se encontra disponível em lojas especializadas, chamados componentes de prateleiras (off-the-shelf). São dispositivos projetados para uso geral, tais como portas NAND, AND, decodificadores, etc, e produzidos em grandes quantidade garantem baixos preços.

Quando é necessário um componente que tenha como requisitos mínimo preço e alta performance o projeto de circuito dedicado é recomendável.

O projeto de um circuito dedicado pode ser implementado de três formas básicas quanto ao estilo de projeto:

- full-custom,
- gate-arrays e
- standard cell.

3.3.1 Estilo Full-Custom (Totalmente Dedicado)

Dentro do estilo full-custom, ou totalmente dedicado, o projetista tem a liberdade de projetar desde as estruturas mais elementares, os transistores, até as mais complexas tais como sistemas e subsistemas. Portanto, dado um determinado processo de fabricação, o projetista determina as dimensões de cada transistor, suas ligações para a formação de portas lógicas, a interligação das portas lógicas constituindo os subsistemas e finalmente, a inteconexão dos subsistemas formando o sistema global.

Como características básicas deste estilo temos a otimização de área e de performance do sistema como um todo, obtida através da otimização das estruturas mais elementares, mas em compensação o ciclo de desenvolvimento de um circuito integrado e' muito mais longo, requerendo ainda um conjunto de ferramentas de projeto mais sofisticado e menos automatizado que os demais estilos. Seu emprego e' justificado quando o produto será produzido em grandes quantidades ou ainda em aplicações onde a otimização da performance e' exigida.

3.3.2 Estilo Semi-Dedicado (Gate-Array e Standard Cell)

As estruturas mais elementares tais como os transistores, portas lógicas ou ainda pequenas funções tais como contadores, somadores, etc, podem ser usadas a partir de uma biblioteca já previamente implementada. A este estilo de projeto chamamos de projeto semi-dedicado. Dentro do estilo de semi-dedicado temos ainda uma outra divisão: gate-arrays e standard cells.

A diferença entre os dois aparece na implementação das funções no silício. Enquanto nos gate-arrays as portas lógicas são implementadas a partir da interligação de transistores padrões já difundidos no silicio, os standard cells comportam-se como pequenos projetos full-custom para cada célula retirada da biblioteca. Não existe nenhum processo pré-difundido no silicio num standard cell assim como no full-custom.

Um fator que representa muito no projeto de semi-dedicados e' o alto nível de automação das ferramentas usadas para este estilo devido a sua regularidade de estrutura. Este nível de automação faz com que o ciclo de desenvolvimento seja menor,

não requerendo um nível de especialização do profissional como o exigido pelo projeto full-custom.

3.4 *Fluxo de Projeto de Circuitos Integrados*

Com a evolução da microeletrônica desde seus primeiros componentes discretos semicondutores ate' os circuitos de alta integração, o projeto de sistemas digitais alterou-se profundamente, passando do projeto de placas de circuito impresso com componentes discretos a projeto de sistemas completos no silício. Entre estes dois extremos, existe uma alteração profunda de metodologias e ferramentas de auxílio de projeto.

O projeto de um chip inclui o projeto do sistema digital e sua implementação a nível de componentes, onde encontramos algumas fases coincidindo com o projeto de uma placa de circuito impresso. O projeto de circuitos integrados porém, e' diferenciado do projeto de sistemas em placas pela sua complexidade, alto custo de implementação e principalmente pelo ciclo de desenvolvimento e teste (turnaround time). As implementações de alterações após o processamento do chip consomem praticamente um novo ciclo completo. Um conjunto de ferramentas de software e' necessário para minimizar este tempo.

Podemos dividir o projeto de um circuito integrado em duas fases:

- projeto lógico;
- projeto fisico.

No projeto lógico e' executada a descrição do circuito desde o projeto funcional até o nível de portas lógicas. O projeto fisico compreende a implementação a nível de transistores e seu layout.

3.4.1 Projeto Lógico

Podemos dividir o projeto lógico em algumas fases distintas:

- projeto funcional;
- projeto de subsistemas;
- implementação em portas lógicas.

A partir do projeto funcional do chip, onde são definidas funções, sinais de entrada e saída, sinais de controle e frequência de clock (caso seja necessário), passa-se ao projeto dos subsistemas que compõem o chip. O projeto de subsistemas consiste na estruturação do projeto através da redução do sistema desejado a blocos funcionais já conhecidos como autômatos, ROM'S, etc. Após o projeto dos subsistemas, a próxima etapa consiste em implementar estes em portas lógicas fundamentais como And, Or, flip-flops, etc.

Para minimizar a probabilidade de erro de projeto e a consequente perda do esforço desenvolvido, torna-se necessário o uso de um conjunto de ferramentas de software que permita simular o circuito de maneira conclusiva quanto a sua funcionalidade. Estes testes devem ser exaustivos e simular condições de funcionamento do circuito em operação real. Esta fase de simulação lógica ou funcional e' executada independentemente da tecnologia que se pretenda adotar e também do estilo para implementação do circuito (full-custom, standard cell ou gate-array). Para uma simulação mais apurada, elementos de atraso deverão ser considerados e não

simplesmente a simulação de chaveamento. Alguns simuladores já consideram atrasos de portas, utilizando dados do processo a ser empregado. No projeto full-custom, estes elementos devem ser considerados pelos projetistas.

Para execução destas etapas são necessárias as seguintes ferramentas:

- editor de esquemático;
- simulador lógico.

O editor de esquemático tem por objetivo a descrição do circuito em termos de blocos lógicos elementares que implementam uma determinada função, gerando o arquivo de entrada para o simulador lógico que, fazendo uso dos vetores de testes, simula o circuito, testando sua funcionalidade. O simulador dedicado (produto de determinada foundry) incorpora atraso de acordo com seu processo de fabricação. Estes valores não dependem exclusivamente do processo, mas também da construção dos transistores, sendo portanto, exclusivo de fabricante. Nesta simulação, devem ser considerados também atrasos decorrentes das interconexões entre os diversos subsistemas. Esta fase deve ser executada após o roteamento. Após a edição dos circuitos lógicos, teremos a simulação do circuito, onde serão confrontados os resultados obtidos com as especificações desejadas.

3.4.2 Projeto Físico

Após a simulação lógica, o circuito já esta' projetado funcionalmente, simulado e descrito em função de transistores ou standard cells, conforme o estilo adotado. A partir do projeto lógico o próximo passo e' a sua implementação no silício, ou seja, descrever em termos de layout de níveis (layers) o projeto lógico definido na etapa anterior. Chamaremos esta fase de projeto físico.

O projeto físico consta de várias etapas:

- edição das células básicas;
- simulação elétrica das células básicas;
- edição dos blocos lógicos;
- simulação elétrica dos blocos lógicos;
- alocação dos blocos lógicos no plano do silício;
- interligação dos blocos lógicos (roteamento);
- extração do circuito;
- simulação elétrica de caminhos críticos;
- verificação de regras de projeto (DRC);
- simulação lógica do circuito total a partir da saída do extrator.

A edição das células básicas e' um esforço necessário para construção de células fundamentais para a construção dos blocos lógicos. Este esforço e' eliminado quando se utiliza o estilo standard cells ou gate-arrays, podendo ainda ser minimizado quando se utiliza biblioteca padrão de células já confeccionadas. O uso de células de biblioteca não elimina por completo esta etapa do projeto, já que e' impossível prever todas as células que se pretende utilizar. A edição de células e' um trabalho que pode ser executado através de terminais gráficos, onde se projeta diretamente a célula, ou ainda através de descrição das mesmas em alguma linguagem de descrição gráfica (LDG), o que aumenta muito o tempo de edição das células. Após a definição de cada célula, constitui-se um conjunto para utilização no projeto, ou seja cada bloco sera' constituído em função daquelas células básicas. A construção dos blocos funcionais em

função das células também pode ser executada em terminais gráficos ou através de uma descrição em LDG.

Após a edição das células, as regras de projeto são verificadas pelo programa verificador de regras de projeto (DRC). Uma nova simulação lógica e' executada para verificação do projeto lógico e posteriormente uma simulação elétrica e' executada para verificação de performance elétrica. Estas simulações são executadas por blocos funcionais isolados a partir da extração do circuito elétrico e seus parâmetros através de descrições gráficas.

Após a obtenção dos blocos funcionais simulados elétrica e logicamente, o próximo passo e' a alocação no plano do silício de todos os blocos funcionais e de suas interligações, definindo o roteamento do circuito (floor-plan).

O roteamento pode ser automático ou manual. Mesmo utilizando roteamento automático uma percentagem do trabalho sera' manual. Este tipo de trabalho e' executado em terminais gráficos.

Novo DRC e' executado após o roteamento e em seguida uma nova extração. Com o floor-plan gerado, uma nova simulação lógica e' executada para verificação global e a simulação elétrica e' utilizada para caminhos críticos, pois exige muito tempo de processamento. Uma comparação entre a descrição do circuito obtida pelo extrator e a fornecida pelo compilador da linguagem de descrição lógica e' realizada. A verificação de regras elétricas e' então executada para evitar erros de consistência elétrica como nós em curtos, nós equivalentes com nomes diferentes, etc .

Após todos estes testes e verificações, o projeto esta' concluído. E' então enviada à fundição (foundry) uma fita magnética contendo os principais arquivos resultantes do projeto tais como o layout físico e os vetores de testes. A fundição executa o processo e testa o componente executando testes de processo, utilizando-se de dispositivos embutidos na pastilha, e testes funcionais através dos vetores de testes que lhe foram entregues juntamente com o arquivo do layout físico.

3.5 Firmware x VLSI

Com o advento do microprocessador no início da década de 70, o projeto de sistemas digitais sofreu profundas modificações e uma série de funções que originariamente eram implementadas por hardware passaram a ser implementadas por firmware dos microprocessadores. A grande vantagem do microprocessador sobre projetos convencionais de hardware era a modularidade e a facilidade de uso dos componentes microprocessadores. Após pouco tempo esta "cultura" de projeto baseada no novo componente estava formada e, mesmo para projetos mais elementares, passou-se a usar microprocessadores. O custo deste componentes caiu muito, fato este que realimentava este demanda. O ferramental para o desenvolvimento de firmware foi amplamente desenvolvido. Datam desta época o aparecimento das estações de desenvolvimento de microprocessadores. A proposta oferecida pelo uso dos microprocessadores em substituição ao hardware convencional era extremamente atraente. Um microprocessador mais elementar substitui cerca de 5000 portas lógicas; se considerarmos apenas o uso da família de TTL 7400 estamos falando da substituição de 1000 CI's de 14 pinos cada. Portanto o uso de microprocessadores como substitutos de hardware convencional popularizou o projeto de sistemas digitais.

Contribuíram para esta utilização dois fatores básicos: primeiro o custo de desenvolvimento de circuitos totalmente dedicados (full-custom) era extremamente alto quando comparado a componentes de prateleiras (off-the-shelf) e o seu ciclo de desenvolvimento era longo e, segundo, o conhecimento e as ferramentas para se desenvolver um circuito dedicado só eram possuídos por grandes firmas de computadores.

O acesso a este tipo de tecnologia era extremamente restrito. Para dar apenas um exemplo, firmas como a DEC - Digital Equipment Corporation, so' passaram a ter circuitos dedicados em seus computadores em 1980. Para isto contribuíram os seguintes

fatores: conhecimento restrito no desenvolvimento de CI's e ferramental quase inexistente, o que tornava o desenvolvimento quase artesenal, reforçando assim a necessidade de pessoal especializado e acesso aos fundidores (foundries) restrito. Nesta época, apenas as gigantes no setor tais como a IBM, Texas e Fairchild possuíam suas próprias foundries para seu uso interno. Com este binômio, custo alto e alto risco, eram poucas as empresas que se aventuravam nesta área.

Na era pré-microprocessador o projetista contava apenas com circuitos off-the-shelf de baixa integração, SSI (Small Scale Integration) ou MSI (Medium Scale Integration); para funções mais complexas, o sistema se tornava fisicamente muito grande gerando problemas tanto de desenvolvimento, como de performance, sem citar manutenção. O sistema baseado em microprocessadores permitiu que uma série de funções anteriormente implementadas por hardware fossem implementadas por firmware, garantindo uma maior modularidade no desenvolvimento e ainda uma redução de custo com o aumento da complexidade de funções por placa física no sistema. Os sistemas passaram então a ser baseados em microprocessadores, sendo que somente as funções de altíssima velocidade eram deixadas ao hardware "wired" como e' comumente chamado. Esta foi a visão quase generalizada de projetos de sistema na década de 70 e início da década de 80.

Este conceito, no entanto, passou a ser abandonado por dois motivos básicos: a facilidade de cópia de equipamentos permitida pela adoção deste método e pelas facilidades que começaram a surgir para o projeto de circuitos dedicados e semi-dedicados.

O primeiro motivo e' claro pois utilizando-se os componentes off-the-shelf, de fácil aquisição, e estando a "inteligência" do circuito contida em ROM's, sua reprodução torna-se fácil. O uso de circuitos dedicados e semi-dedicados em produtos eletrônicos dificulta o processo de engenharia reversa, apesar de não eliminá-lo como um todo.

Já a facilidade de projeto veio do aparecimento de várias foundries que passaram a processar circuitos projetados externamente a suas companhias, ou seja o processo passou a ser oferecido como um serviço. Portanto, a companhia interessada em projetar um CI já não precisava mais ser especializada também na confecção do mesmo. Juntando-se a isso, tivemos uma popularização do uso de ferramentas de projeto através da criação de pequenas empresas que desenvolviam essas ferramentas.

Nos Estados Unidos a popularização do projeto de CI's começou no final da década de 80. Inicialmente surgiram os cursos ministrados nas universidades, criando-se assim mão-de-obra especializada e, posteriormente, a criação de uma rede de serviços que permitia o acesso aos recursos das foundries. Portanto, qualquer idéia poderia ser implementada no silício a custos muito baixos.

Os Estados Unidos possuem uma rede de serviços para o processamento de circuitos integrados chamada MOSIS, que permite que os circuitos integrados sejam implementados a custos baixíssimos. Isto possibilitou que o programa de treinamento das universidades americanas fosse melhorado muito, e os projetistas passassem a sair de seus cursos superiores já treinados, situação que não acontecia anteriormente, pois o único recurso disponível eram os cursos internos das empresas.

O trabalho desenvolvido por Fuchs e Poulton, que iremos descrever no próximo capítulo, e' um exemplo típico do que foi descrito. Este trabalho teve sua origem em um curso de Introdução aos Sistemas VLSI proferido na Universidade da Carolina do Norte, em 1981.

4.0 Implementação do Algoritmo Z-buffer em Hardware

4.1 Introdução

No capítulo 2 deste trabalho analisamos vários algoritmos que têm como função determinar a visibilidade de objetos de uma cena num sistema gráfico. Estes algoritmos são na sua maioria implementados por software e geralmente executados previamente à exibição da imagem.

Henry Fuchs e John Poulton, ambos da Universidade da Carolina do Norte, apresentaram uma solução em hardware do algoritmo z-buffer. Esta solução foi originariamente apresentada em 1981 através de um artigo na revista VLSI Design /FUCH81/ , hoje High Performance Systems Design e propunha um circuito integrado dedicado full-custom para executar a comparação pixel a pixel da coordenada z de cada polígono: o processador de pixels.

O processador de pixels e' capaz de executar rapidamente o cálculo de linhas e superfícies escondidas a partir de uma estrutura de dados de polígonos e posteriormente passar seus resultados para o controlador de vídeo de dispositivos raster. Este processador de pixels tem como principal característica a velocidade de processamento através de hardware especial que atua em cada pixel da tela. Cada processador calcula a visibilidade correspondente a seu pixel, utilizando-se de uma arquitetura distribuída. Portanto, o cálculo de visibilidade de toda a imagem e' executado paralelamente, pixel a pixel, a cada novo polígono processado. As demais tarefas, tais como tranformações de coordenadas, recorte, etc, são executadas pelo sistema gráfico no sistema hospedeiro.

Como os cálculos de visibilidade são executados pixel a pixel em cada polígono o processamento e' independente do tamanho dos polígonos bem como da sua orientação.

Um sistema de síntese de imagens pode ser descrito segundo o diagrama apresentado na Figura 16 na página 34:

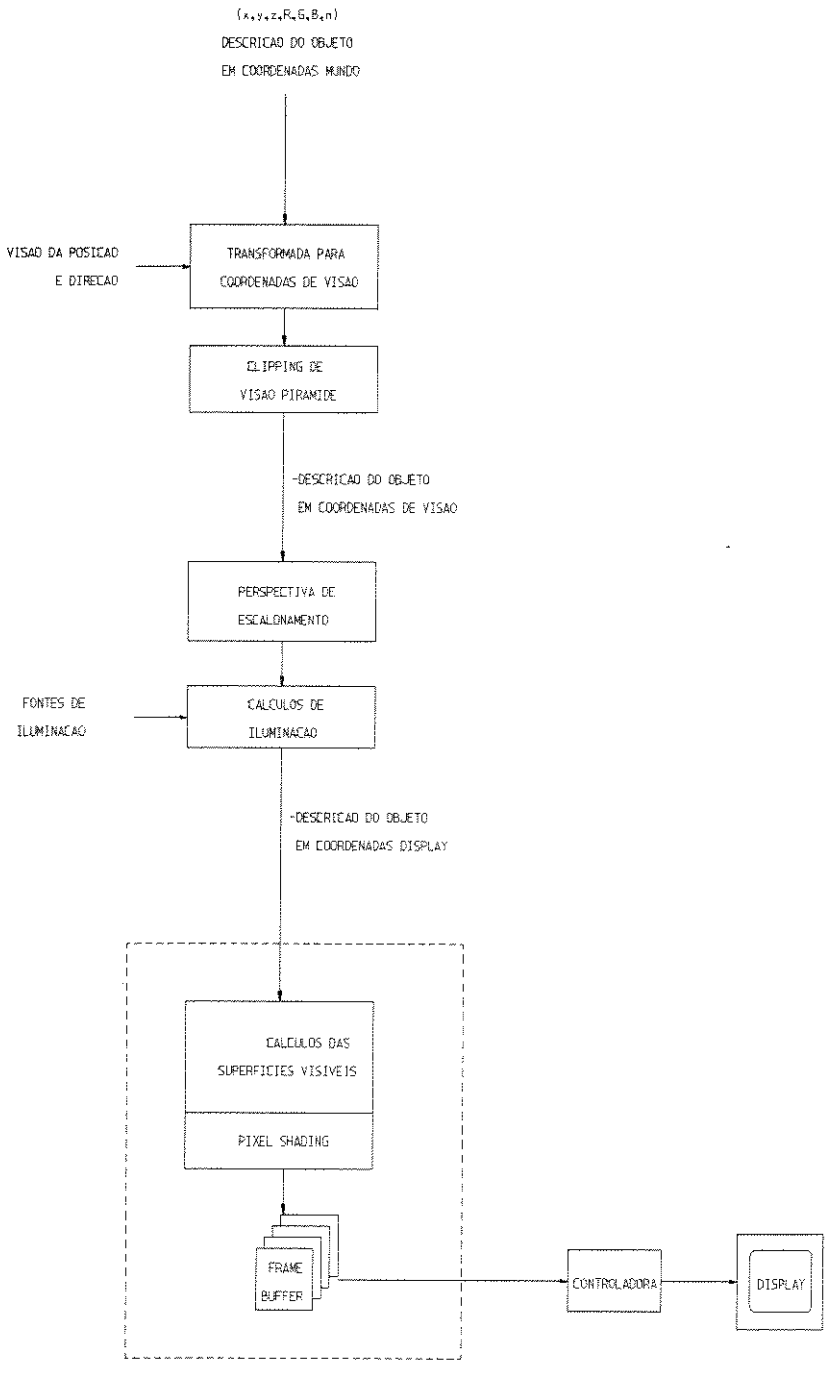


Figura 16. Sistema de síntese de imagens

A estrutura de dados contém a descrição da cena constituída de poliedros, cada qual sendo descrito por um conjunto de polígonos que são aproximações de suas faces. Este polígonos são processados um a um numa determinada ordem. Cada polígono e' descrito por uma sequência de vértices cujas coordenadas x, y e z são expressas no

sistema de coordenadas "mundo". Cada vértice tem também associado a informação de cor. O primeiro processamento executado sobre o polígono e' a transformação de suas coordenadas do sistema mundo para o sistema da visualização, de acordo com a posição do observador. Então e' recortado segundo a pirâmide de visualização, eliminado-se assim aquelas partes do polígono que estão fora do campo de visão. Após este processo o polígono e' escalonado para a perspectiva do dispositivo, expressando assim suas coordenadas no sistema de tela. Os cálculos de iluminação de reflexão e refração são efetuados levando-se em conta as fontes de iluminação diretas e indiretas da cena.

A próxima tarefa do sistema consiste no cálculo da visibilidade dos objetos, que descreveremos no próximo tópico, como parte da apresentação do processador de pixels. No entanto, dois pontos devem ser ressaltados: primeiro, que ate' esta tarefa o número de cálculos efetuados e' dependente do número de polígonos e que o sistema de cálculo de visibilidade (SV) tem como entrada a descrição dos objetos no sistema de coordenadas do dispositivo,

4.2 Descrição Geral do Sistema de Visibilidade (SV)

O sistema de visibilidade (SV) e'constituído de duas partes:

- um pré-processador que converte os dados dos polígonos ja' descritos em coordenadas do displ _ para um formato adequado ao circuito integrado;
- um conjunto de circuitos integrados que executam os cálculos de visibilidade: os processadores de pixels.

O SV emprega o algoritmo z-buffer como base para os cálculos de visibilidade.

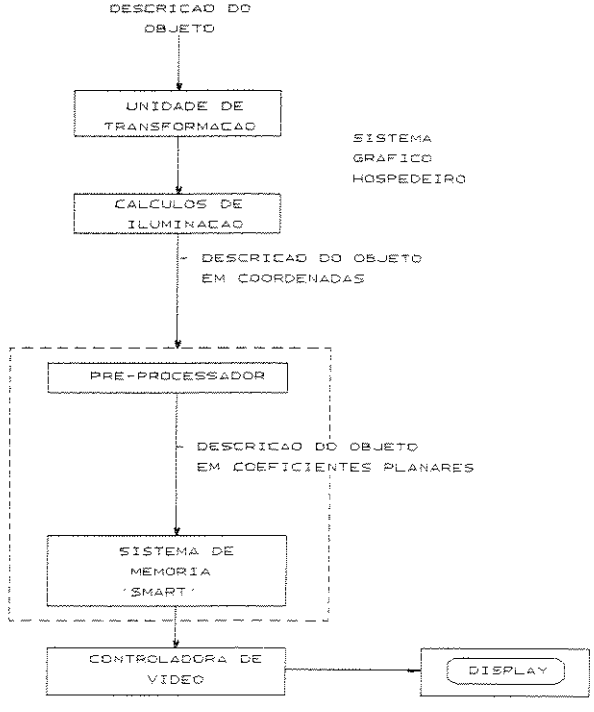


Figura 17. Sistema de visibilidade

4.2.1 Pre'-processador

O pré-processador recebe os vértices dos polígonos para que seja efetuada a primeira operação: a triangularizacão dos mesmos. Os vários vértices são fornecidos para que seja efetuado o cálculo dos coeficientes na forma da equação abaixo:

$$F(x,y) = Ax + By + C \text{ onde } (x,y) \text{ representam cada pixel.}$$

Cada triângulo e' codificado em três vetores com a representação acima, que nada mais e' que a equação de uma reta. Portanto para cada triângulo teremos três vetores: o primeiro do vértice V1 ate' o V2, o segundo do vértice V2 para V3 e o terceiro do vértice V3 ate' o V1, fechando assim o triângulo. Cada um destes vértices deve ser fornecido para o cálculo dos coeficientes numa ordem pre'-determinada horária ou anti-horária, mas devendo esta ordem ser mantida do primeiro ate' o ultimo polígono.

A equação da reta entre os pontos Vi(Xi,Yi) e Vi + 1(Xi + 1,Yi + 1) e':

$$(Y-Yi) = \frac{(Yi+1 - Yi)\ (X + Xi)}{(Xi+1 - Xi)}$$

reagrupando teremos:

$$-(Yi+1-Yi)X + (Xi+1-Xi)Y + (Yi+1-Yi)Xi - (Xi+1-Xi)Yi = 0$$

Comparando a equação acima com F(x,y) temos:

$$A = -(Yi+1-Yi) \text{ e} \qquad B = (Xi+1-Xi) \text{ e} \qquad C = -(Xi*A + Yi*B)$$

Para cada vetor que constitui um triângulo e' codificado um conjunto de coeficientes A,B, C' e C'' onde C = C' + C'' como podemos observar na Figura 18 na página 37

Com a substituição de (x,y) para o pixel atual nas 3 equações Ax + By + C = 0 para os vetores V1,V2; V2,V3; V3,V1, obteremos, dependendo da convenção, se o ponto esta' 'a direita ou 'a esquerda do segmento da reta.

Assim, se F(x,y) para todos os vetores que compõem o triângulo for não negativo o ponto (x,y) esta' no interior do triângulo (quando os vetores são codificados no sentido horário, o contrário no sentido anti-horário). Então o pixel e' interior ao polígono.

A coordenada z de cada pixel e' codificada em outro conjunto de coeficientes A, B, C segundo a equação planar .

$$z(x,y) = F(x,y)$$

Como podemos observar na Figura 18 na página 37, o pré-processador converte as informações de cada vértice em coeficientes necessários à resolução da equação apresentada acima. Assumimos que estas informações provenientes do hospedeiro são enviadas numa determinada ordem e que o último vértice não sera' transmitido pois ele e' o mesmo que o primeiro. Primeiramente o pré-processador divide um polígono de n lados em n-2 triângulos, sendo que cada um partilha com o previamente processado um vértice comum; so' após isso os coeficientes das equações planares são calculados.

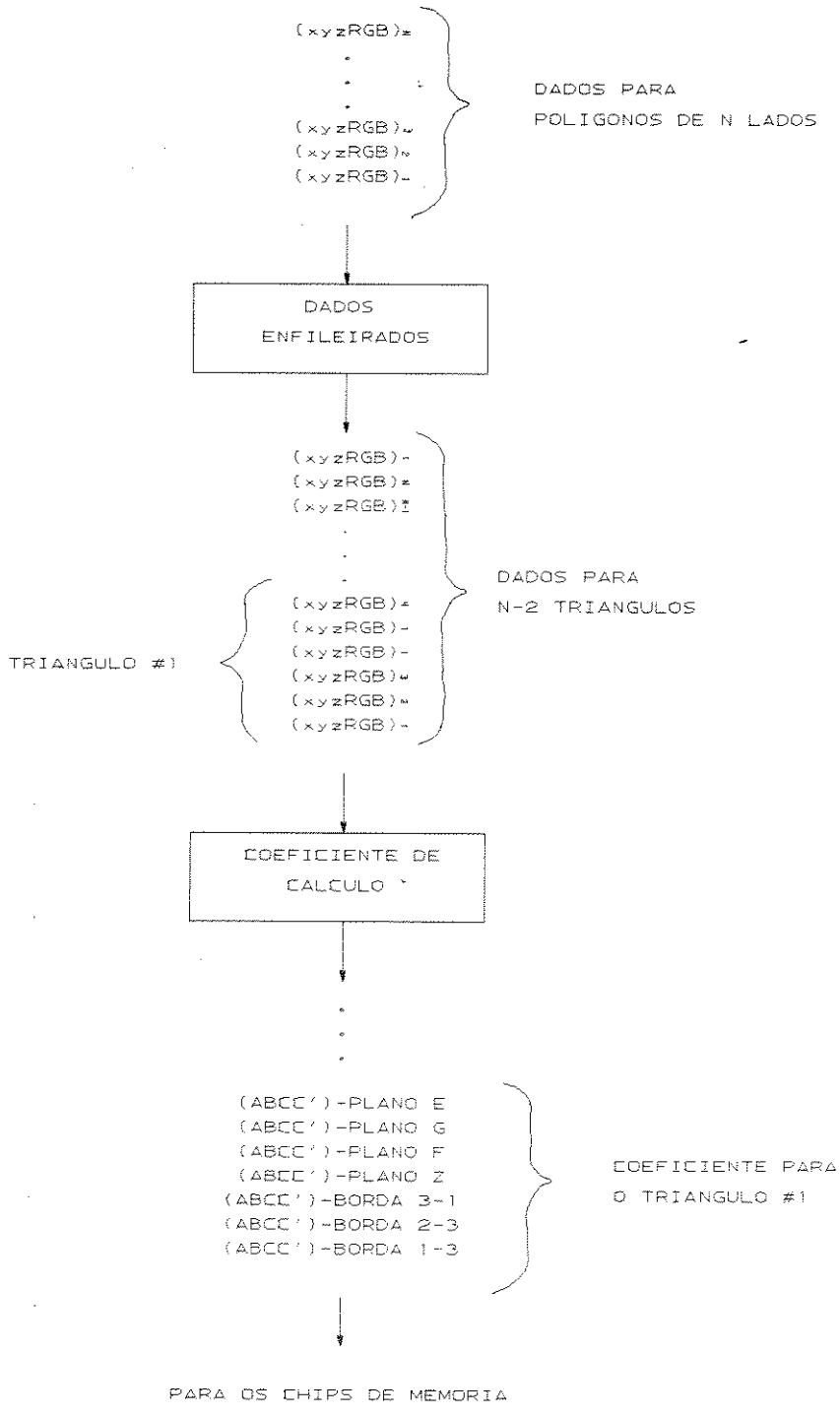


Figura 18. Fluxo de informações no pre'-processador

4.2.2 Processador de Pixels

O processador de pixels e' constituído por multiplicadores X e Y e memórias de pixels.

As memórias de pixel são interligadas constituindo uma matriz como podemos ver na Figura 19 na página 38 e as informações são alimentadas através de barramentos.

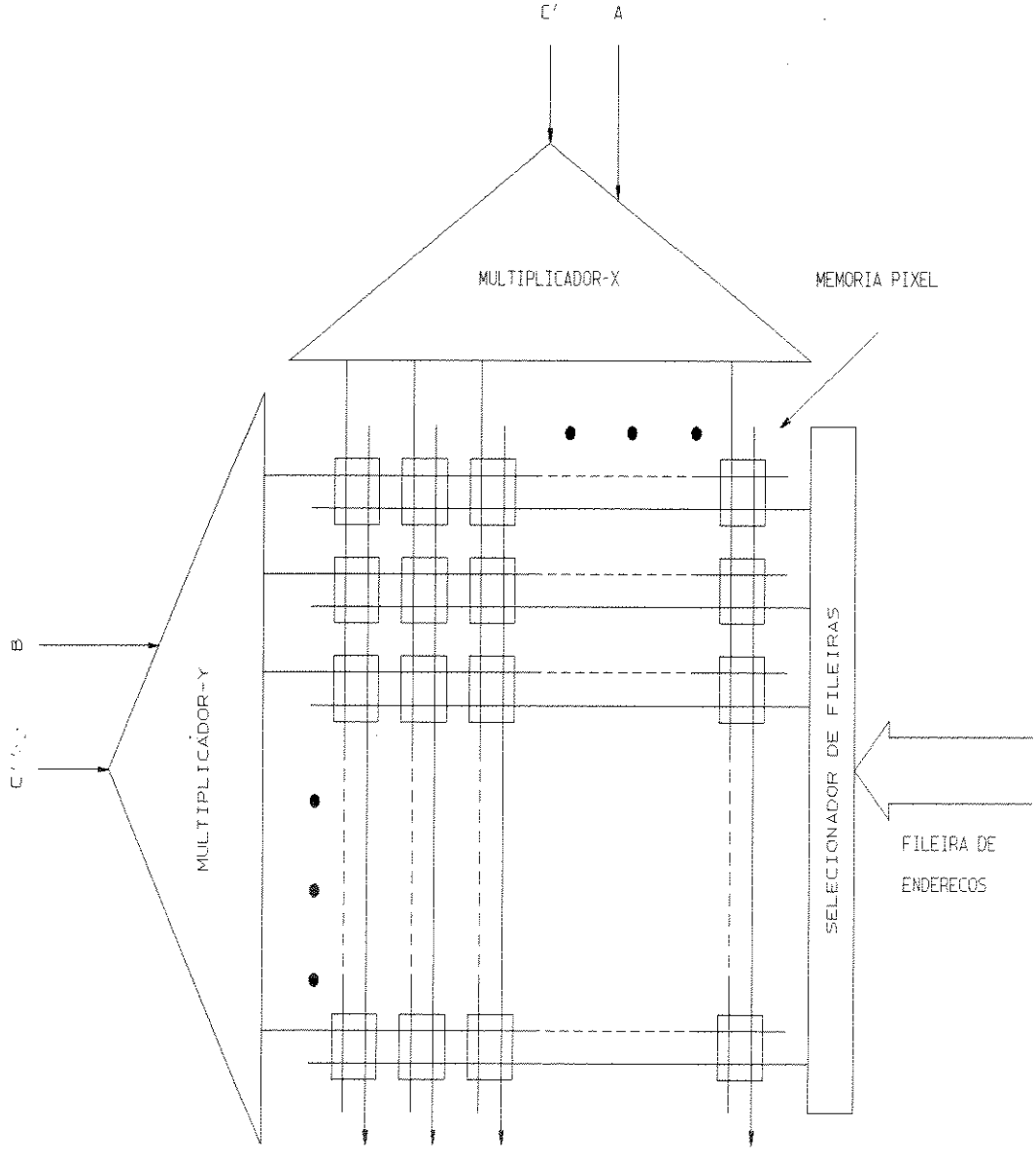


Figura 19. Processador de pixels: multiplicadores (X e Y) e memórias

Dois multiplicadores (X e Y) são responsáveis pelo cálculo da equação F(x,y) para todo pixel (x,y) da tela. Estes têm como entradas os coeficientes A, B e C do pre'-processador e calculam simultanemente, para todos os valores de x e y da tela, as funções $Ax + C'$ e $By + C''$, onde $C = C' + C''$.

Uma representação serial destes parâmetros (A, B, C′ e C″) alimenta os multiplicadores.

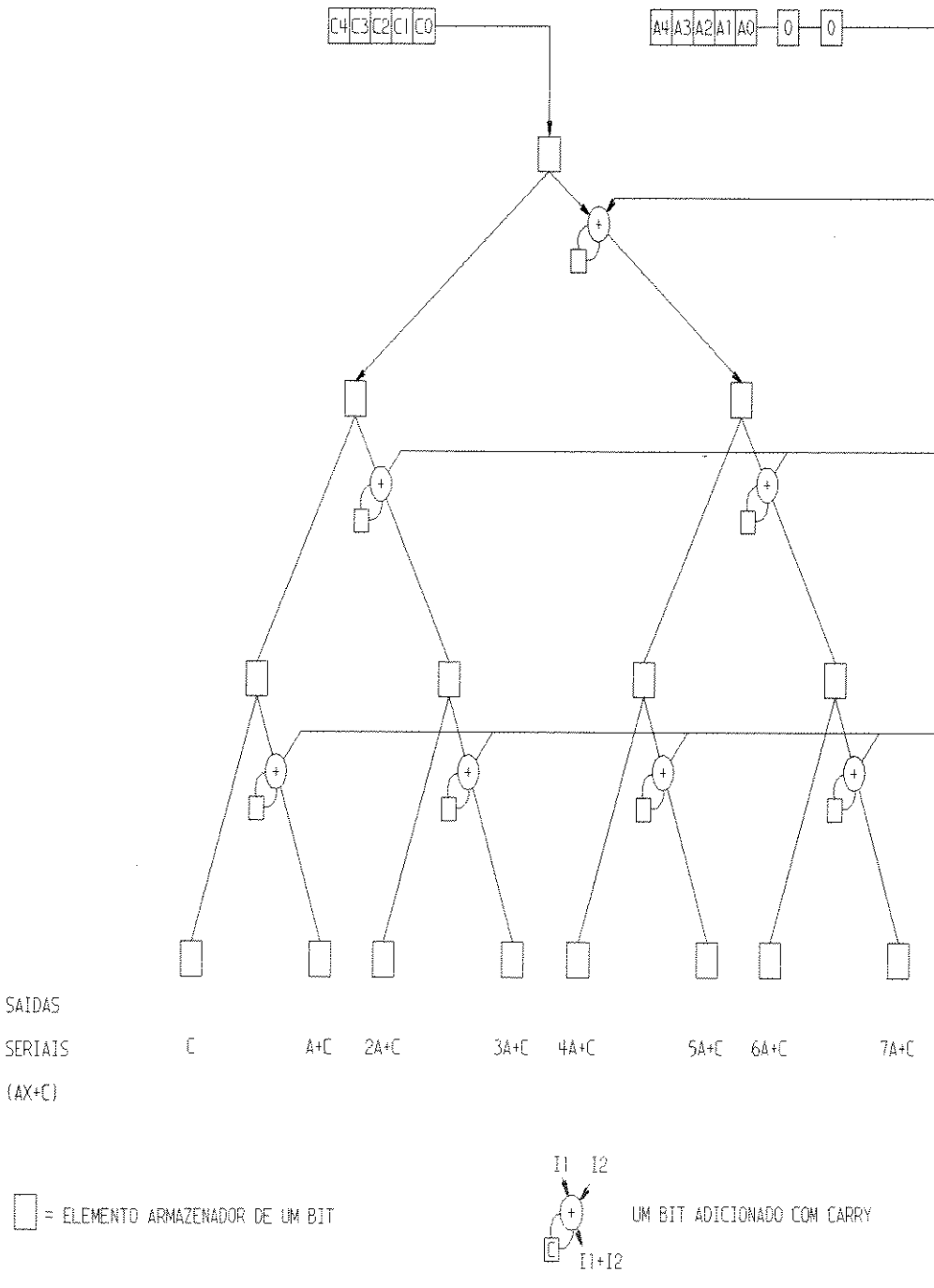


Figura 20. Multiplicador X (análogo a Y)

A representação serial da função ja′ calculada e′ alimentada junto às memórias de pixel. Assim para o multiplicador X temos as saídas:

- C′
- A + C′
- 2A + C′
- 3A + C′
- 4A + C′
- e assim sucessivamente.

E para o multiplicador de Y:

- C″
- B + C″
- 2B + C″
- 3B + C″
- 4B + C″
- e assim sucessivamente.

Estas operações são realizadas de forma serial, a partir de uma cadeia de bits de entrada (A, B, C′ e C″) que geram os resultados acima nas respectivas saídas.

Portanto para cada termo da matriz (x,y) de pixels, temos o valor Ax + C′ e By + C″ correspondente.

Para melhor entendimento do funcionamento do multiplicador vamos tomar como exemplo que A seja um número binario de quatro bits:

A3 A2 A1 A0

Tal como C:

C3 C2 C1 C0

A função 4A + C pode ser calculada:

```
   A3 A2 A1 A0 x
   0 1 0 0
   ----------------
A3 A2 A1 A0 0 0  +  C3 C2 C1 C0  =
```

o resultado:

MSB A3 A2 (A1 + C3) (A0 + C2) C1 C0 LSB

< ---------- t

MSB-bit mais significativo
LSB-bit menos significativo
Portanto, numa representação serial do resultado acima teremos na saída do multiplicador correspondente a 4A + C, considerando que dois zeros são introduzidos em t0 e t1:

tempo	saída (4A + C)
t3	C0
t4	C1
t5	o resultado de (A0 + C2)
t6	o resultado de (A1 + C3)

t7 A2
t8 A3

Para a saída (7A + C) do multiplicador teremos:

tempo saída (7A + C)
t3 o resultado de (A0 + C0)
t4 o resultado de (A0 + A1 + C1)
t5 o resultado de (A0 + A1 + A2 + C2)
t6 e assim sucessivamente.

Os demais resultados podem ser encontrados repetindo as operações que foram apresentadas acima.

4.2.3 Operação do SV

Na Figura 21 e' apresentado um esquema conceitual de uma célula de memória do pixel:

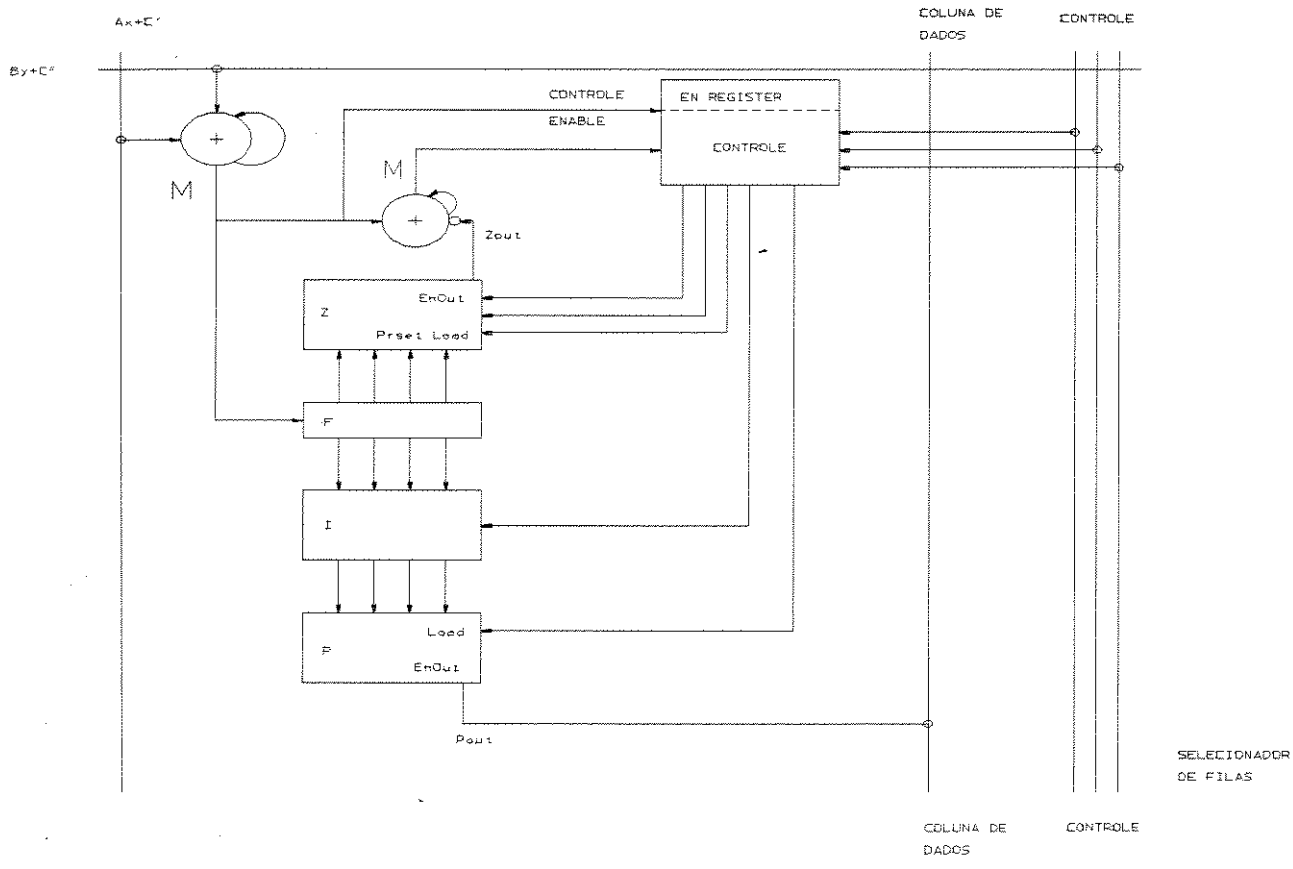


Figura 21. Modelo conceitual de uma célula de memória de pixel

Como podemos observar a memória de pixel contém 5 registros:

- registro Z, que contém o menor valor de z recebido ate' o momento naquele pixel;
- registro F, que contém o resultado da função $F(x,y) = Ax + By + C' + C''$
- registro I, que contém o resultado da operação z-buffer (comparação de Z com F), que posteriormente tera' seu valor armazenado em P;
- registro P, que contém a intensidade da imagem anteriormente construída, ou seja, a intensidade que esta' sendo mostrada no momento. É neste registro que sera' lido o valor de intensidade daquele pixel pelo controlador de vídeo, podendo ser acessado diretamente e independentemente da operação que esta' sendo realizada na célula;
- registro Enable (En) que permite as operações de carga e limpeza dos demais registros desde que F(x,y), resultado do somador M, seja não negativo.

O registro P contém assim o valor da intensidade do pixel (preto/ branco) ou ainda os valores RGB do mesmo.

Operação do SV

1. Para iniciar uma nova cena, todos os registros Z são colocados em valor "1", que significa profundidade infinita.

2. Quando os polígonos começam a ser processados, o registro Enable e' setado, permitindo assim a operação da célula.

3. Cada polígono e' submetido à seguinte sequência de operações:

   a. Cada polígono e' pre'-processado de maneira que cada uma de suas faces seja codificada em coeficientes de F(x,y) tal que se (x,y) reside dentro de seus limites o valor de F(x,y) e' não-negativo ($>=0$) e negativo se o pixel estiver fora de seus limites.

      Os valores são então transmitidos para os multiplicadores que executam as multiplicações. Estes resultados são transmitidos para as células de memória do pixel através dos barramentos. A soma final $Ax+C'$ e $By+C''$ e' executada no somador completo e posteriormente seu resultado e' armazenado no registro F. Caso o sinal seja negativo, o registro En e' desabilitado e toda a célula e' desabilitada ate' o próximo polígono, pois este pixel não compõe o atual polígono.

   b. Caso o valor de F seja positivo, uma comparação entre F e Z e' realizada. Caso $F >= Z$, a porção do polígono esta' escondida por polígonos previamente computados, então o registro En e' desabilitado e não havera' mais nenhum processamento. O valor de Z permanece o mesmo e o polígono que estava sendo mostrado continua o válido para aquele pixel. Caso $F < Z$, o polígono que esta' sendo processado esta' mais próximo do observador do que o previamente processado, portanto ele e' visível naquele pixel e devera' ser mostrado com sua intensidade. O valor de F então e' carregado em Z, atualizando assim o z-buffer.

   c. Os valores da intensidade do polígono estão expressos na forma $I=Ax+By+C'+C''$, sendo I sucessivamente R, G e B. Estes valores são armazenados no registros I nas sua porções correspondentes.

4. Quando todos os polígonos da cena ja' foram processados os valores de I são carregados em P, estando assim disponíveis para o controlador de vídeo.

No próximo capítulo apresentaremos a implementação da proposta apresentada neste capítulo utilizando a técnica de gate-array.

5.0 Proposta de Implementação em Gate-Array

5.1 Introdução

No capítulo 4 apresentamos a implementação do algoritmo z-buffer proposto por /Fuch81/. Esta proposta apresentada em 1981 foi implementada em um circuito integrado através do estilo full-custom. Neste capítulo apresentamos a implementação da proposta de Henry Fuchs e John Poulton em gate-array de tecnologia CMOS 1.5 mícrons.

Este projeto foi desenvolvido através do sistema de projeto de circuitos integrados existente no Centro de Tecnologia de Hardware da IBM em Sumare', denominado EDS (Engineering Design System). O circuito foi implementado e simulado exaustivamente com as técnicas de projeto que foram apresentadas no capítulo 3 deste trabalho. Os resultados das simulações bem como as listagens dos diagramas lógicos são apresentados no anexo.

5.2 Descrição Geral do Projeto

A Figura 22 na página 46 apresenta o diagrama de blocos funcionais do circuito do processador de pixels. Como apresentado no capítulo 4, o SV (sistema de visibilidade) e' composto pelo pre'-processador e pelo processador de pixels. O projeto do circuito integrado, aqui discutido, so' contempla o processador de pixels, que e' formado pelos multiplicadores X e Y, uma matriz de 8 por 8 unidades aritméticas e lógicas (UAL's) e 64 conjuntos de registros Z, F, I e P que constituem as memórias de pixels. Estes subsistemas são controlados através de uma unidade de controle.

Cada processador de pixel proposto tem a capacidade de controlar 64 pixels e tem como entrada os coeficientes da equação:

$$F(x,y) = Ax + By + C,$$

que são alimentados pelo pre'-processador através de uma representação serial dos coeficientes A, B , C' e C'' onde $C = C' + C''$.

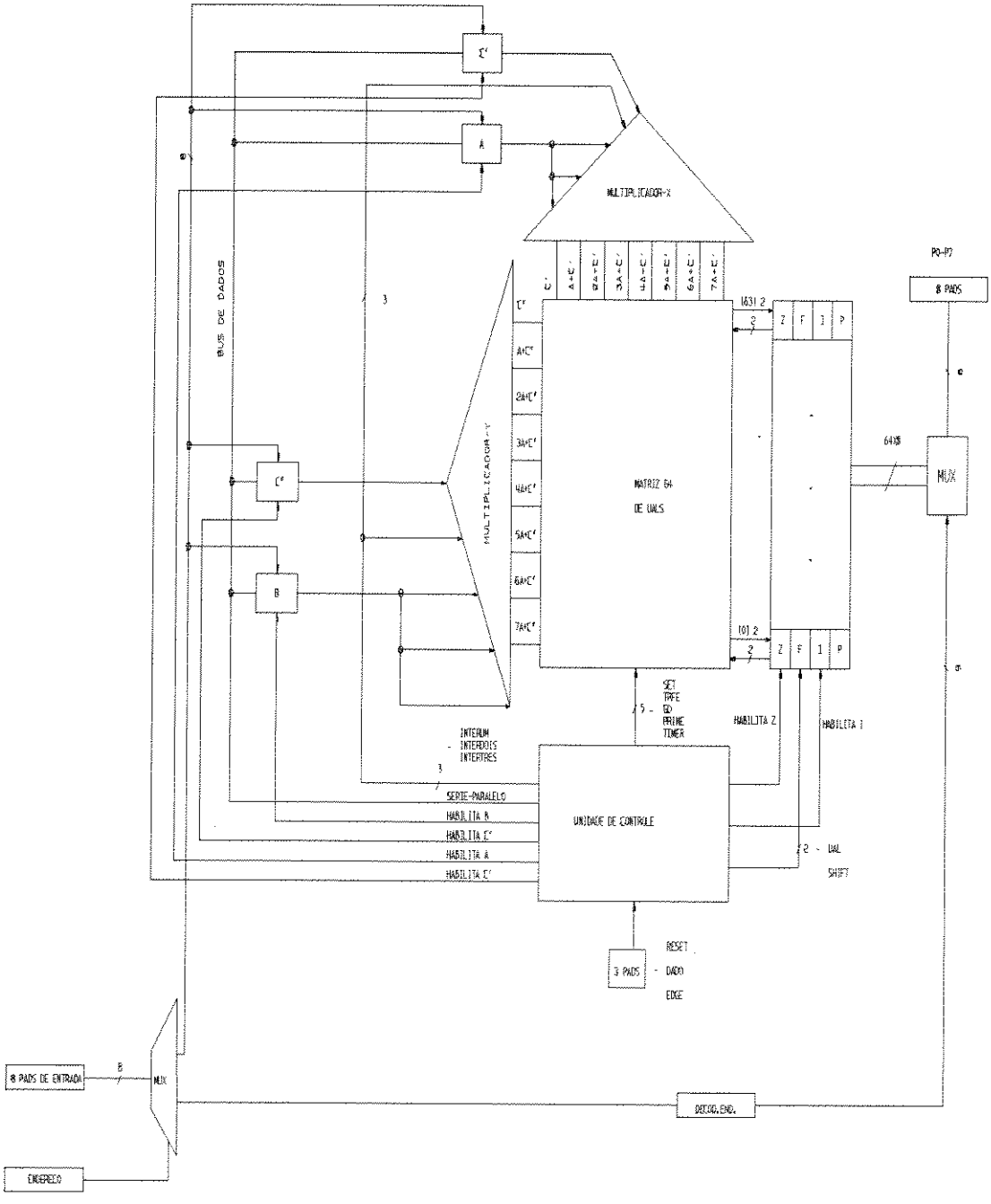


Figura 22. Diagrama geral do processador de pixels

Estes coeficientes são utilizados pelos multiplicadores para calcular o valor da função F(x,y) em cada pixel. Estes valores são alimentados junto às UAL's que calculam o valor final da função para sua posição correspondente. As UAL's têm a si associado os registros Z, F, I e P que contêm os valores de Z naquele pixel desde o último

processamento de polígono, o valor da função F e a intensidade do pixel (I) que esta' sendo mostrada naquele momento, respectivamente.

Quando o valor de F e' calculado, o pixel pode estar contido no polígono que esta' sendo processado ($F >= 0$) ou estar fora dele ($F < 0$). Caso o pixel esteja contido no polígono sua profundidade ($z = F(x,y)$) e' comparada com a profundidade do polígono mais próximo do observador naquele ponto (conteúdo de Z). Caso o novo polígono seja mais próximo do observador que o anterior, o valor de Z e' atualizado e sua intensidade e' carregada no registro de intensidade I. Caso contrário o processador de pixels passa a processar um novo polígono.

As saídas do processador podem ser lidas pelo controlador de vídeo através dos registros P (0 a 63) que contêm o valor da intensidade de cada pixel processado pelo processador de pixels.

5.2.1 Multiplicadores

Este módulo recebe os dados seriais de A e C' (multiplicador X) e B e C'' (multiplicador Y) como entradas e produz 8 linhas de saídas ($Ax + C'$e $By + C''$). Elas representam 8 diferentes valores para a função F(x,y), onde (x,y) são as coordenadas do pixel.

A célula básica do multiplicador e' apresentada na Figura 23 na página 48:

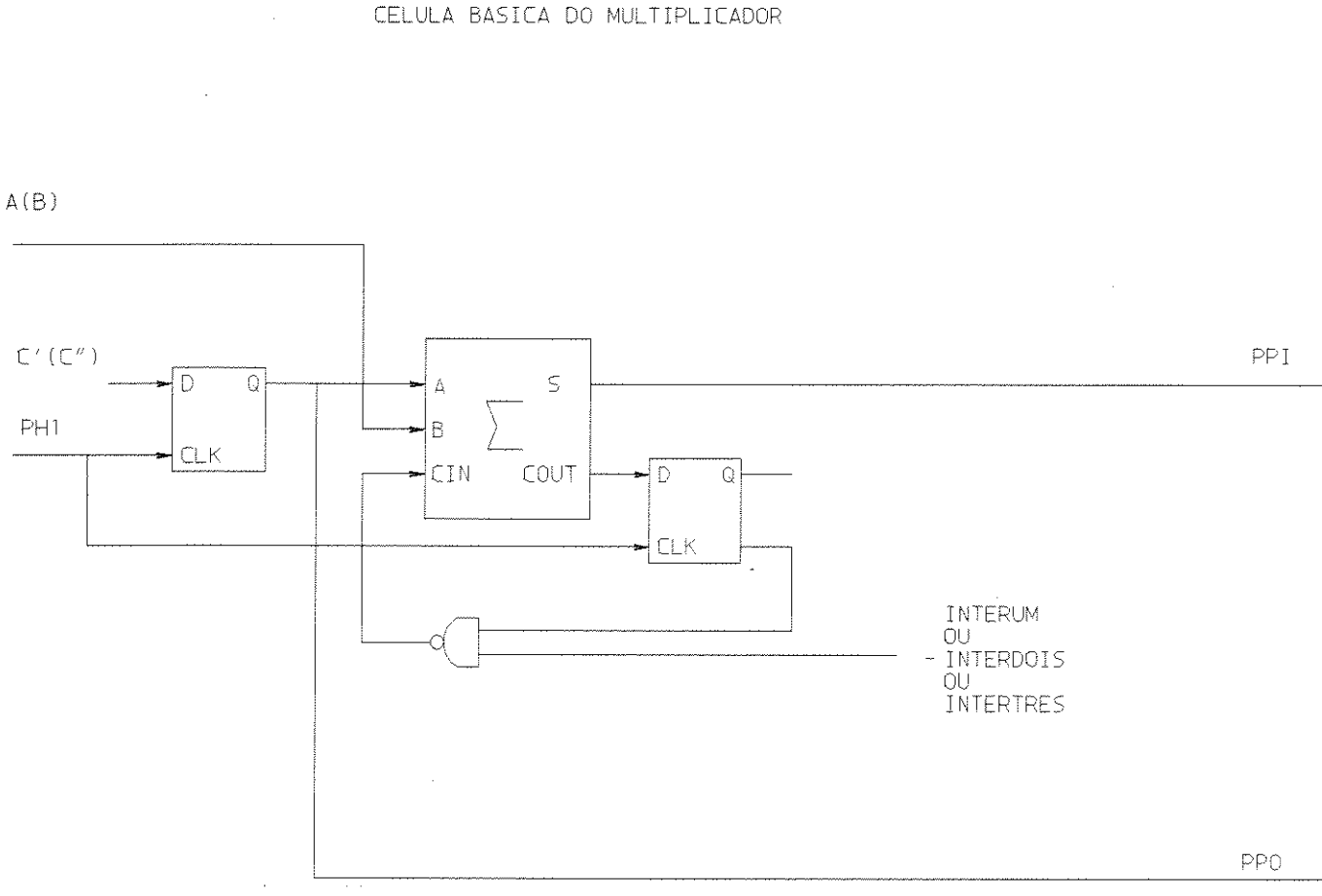


Figura 23. Célula básica do multiplicador

O multiplicador executa sua função serialmente; o resultado e' fornecido do bit menos significativo (LSB) ate' o bit mais significativo (MSB).

5.2.2 Unidade Aritmética e Lógica

A unidade aritmética e lógica (UAL) do processador de pixels e' apresentada na Figura 24 na página 49.

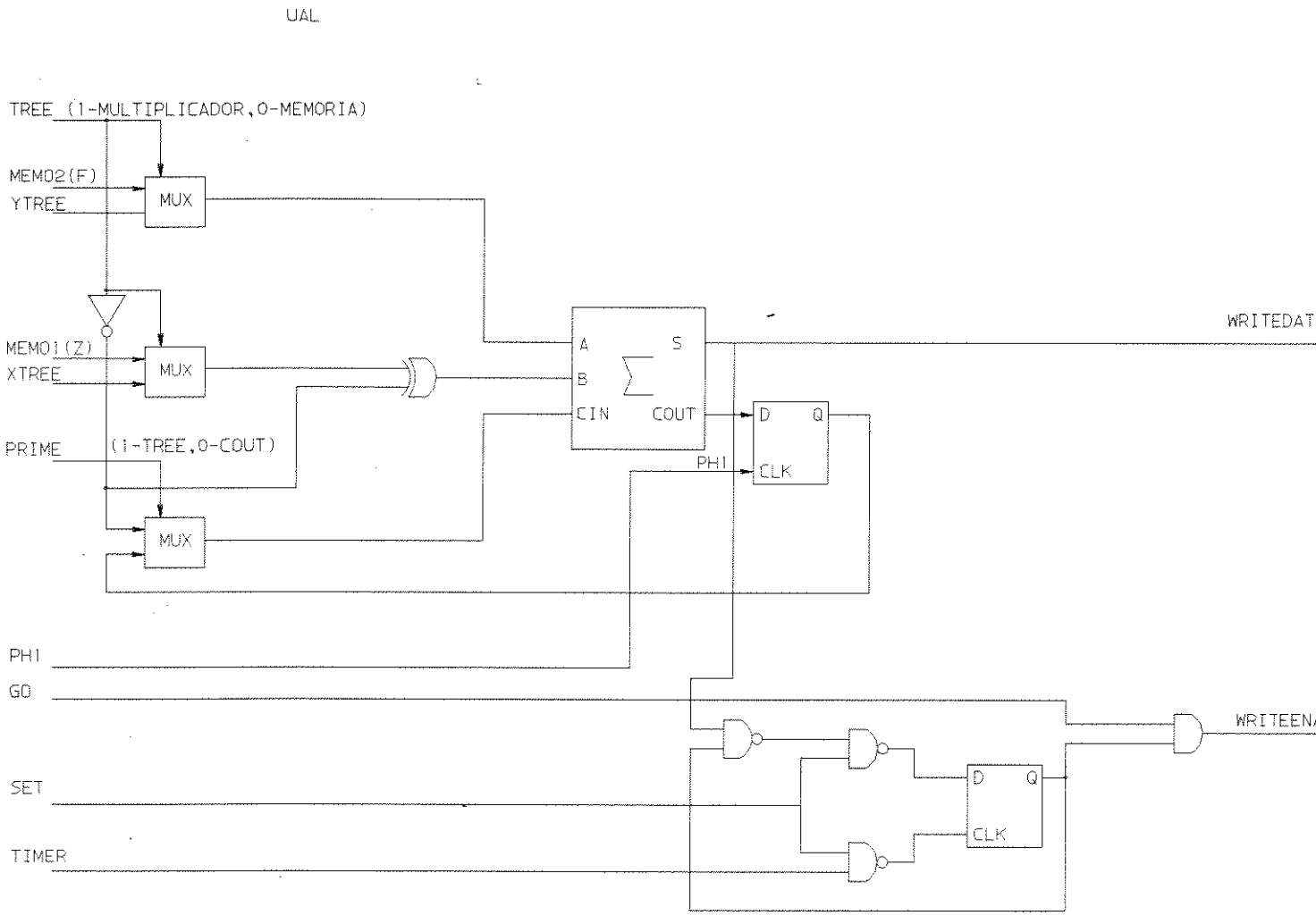


Figura 24. Célula básica da UAL

Esta UAL permite a seleção entre a soma dos resultados do multiplicador bem como a soma dos conteúdos dos registros Z e F, através do sinal de controle UAL. A UAL executa portanto a soma das funções $Ax+C'$ e $By+C''$ para verificar se ela e' não negativa e também a comparação bit a bit dos registros Z e F. Diferentemente da apresentada no capítulo 4, esta UAL so' tem um somador completo, o que permite uma melhor utilização do circuito com uma minimização de área no circuito integrado, uma preocupação constante de todo projetista.

O resultado do somador completo controla o sinal WRITE ENABLE, que habilita a escrita nos registros Z e F, pois caso F(x,y) seja negativa os valores dos registros Z e F não serão alterados. Caso contrário, o valor de F e' carregado para posterior comparação bit a bit com Z.

Para comparação de Z com F, o sinal UAL seleciona os registros Z e F e passa a comparar o complemento de Z com F. Assim se o resultado de F-Z e' positivo ou nulo o conteúdo dos registros não e' alterado, caso contrário o valor de F e' carregado em Z e este valor passar a ser o menor valor de Z naquele pixel.

5.2.3 Unidade de Controle (UC)

A unidade de controle gera todos os sinais de controle necessários para o funcionamento do processador de pixels, atuando nos multiplicadores, UAL's e memórias de pixels, como podemos observar na Figura 25 na página 51.

A unidade de controle e' composta por um sequenciador e um decodificador de estados. O sequenciador e' uma máquina de estados que determina qual operação esta' sendo realizada pelo processador de pixels: carga dos coeficientes, multiplicação, comparação, etc. O decodificador decodifica este estado e gera os sinais de controle apropriados, que veremos a seguir.

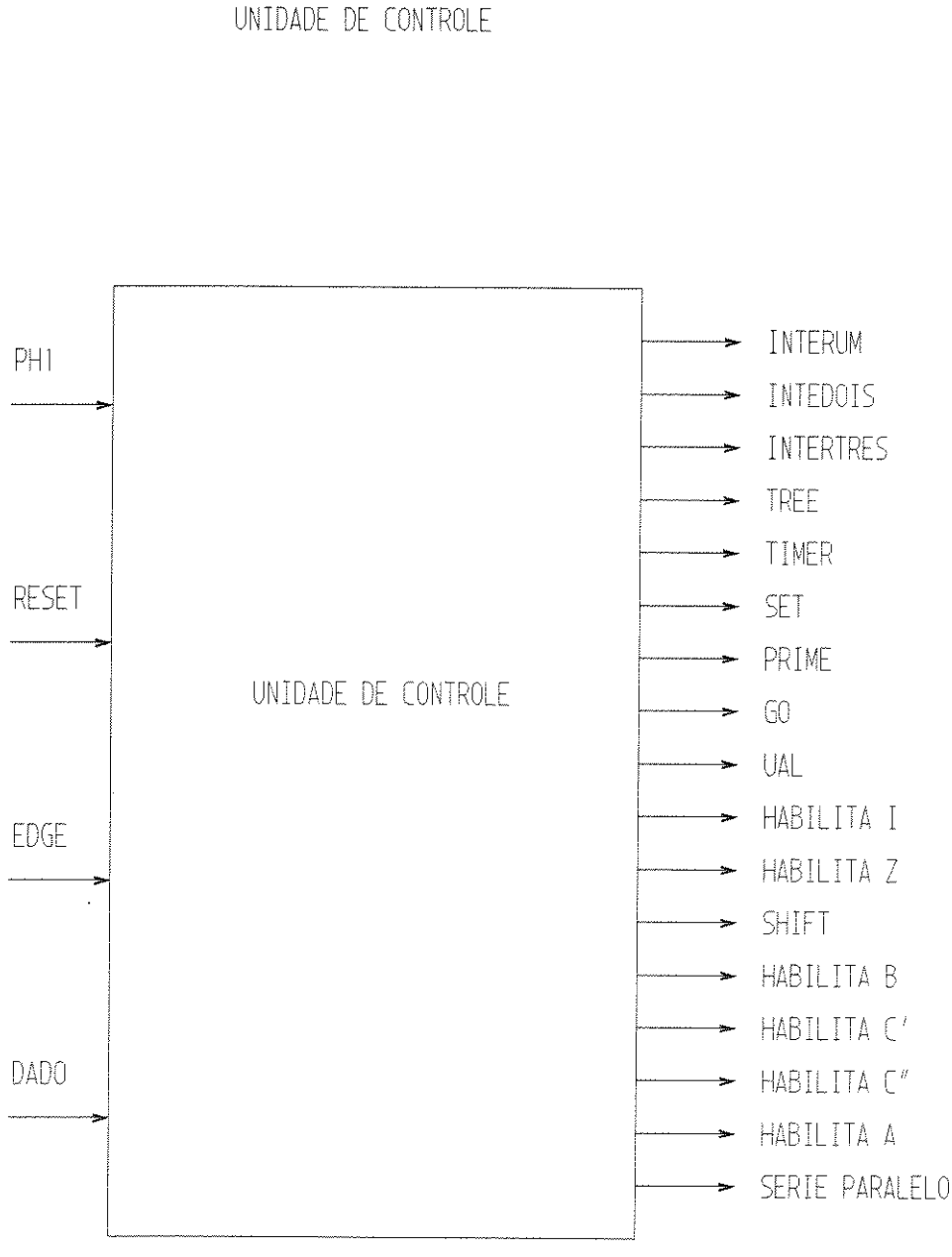


Figura 25. Unidade de Controle

A UC tem como sinais de entrada:

- Ph1, que e' o clock do sequenciador;
- Reset, que tem como função reinicializar o sistema;

- Edge, indica que uma nova aresta do polígono esta sendo processada;
- Dado, indica que um dos coeficientes A, B, C′ ou C″ esta′ disponível na entrada;

A unidade de controle gera como sinais de saída para os registros dos coeficientes (Figura 26 na página 53):

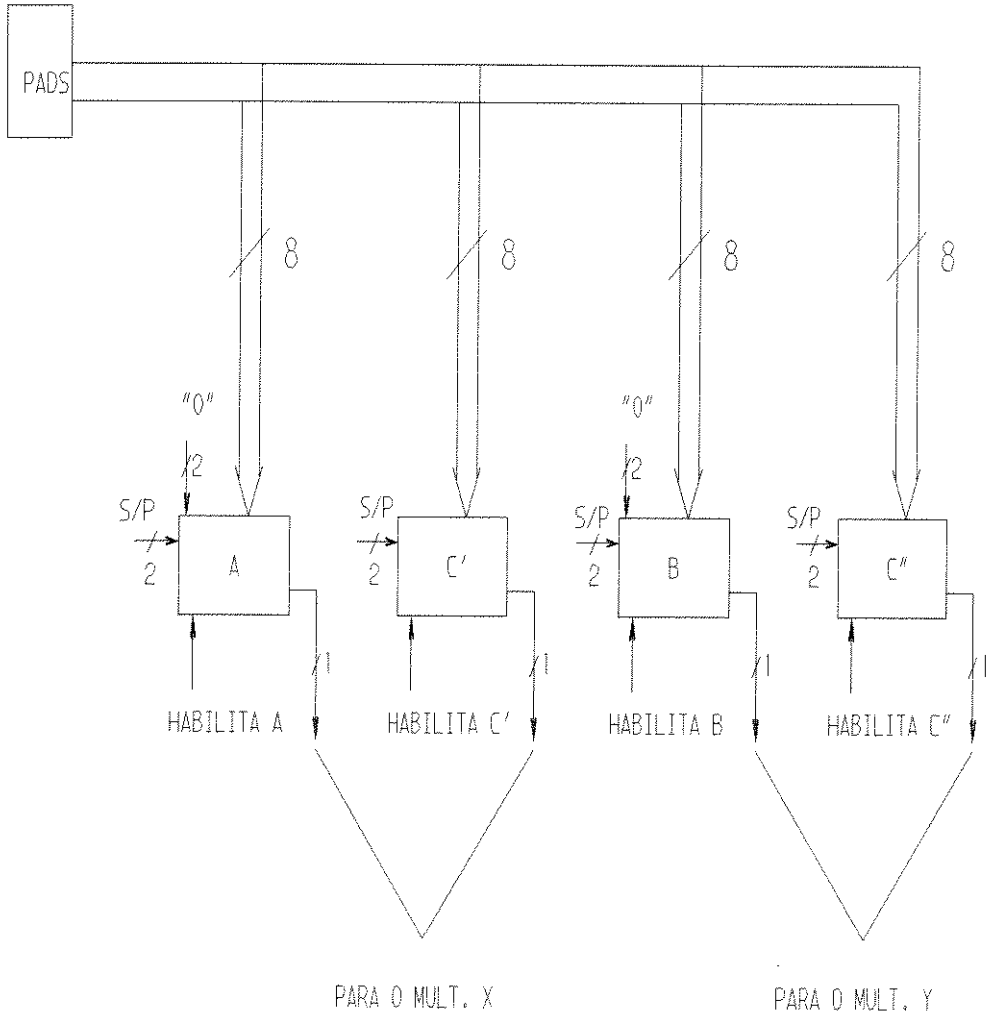


Figura 26. Registro dos coeficientes A, B, C′ e C″

- Habilita A, que habilita o registro A para carga do valor,
- Habilita B, que habilita o registro B para carga do valor,
- Habilita C′, que habilita o registro C′ para carga do valor,

- Habilita C″, que habilita o o registro C″ para carga do valor,
- Série, quando este sinal está com valor '1', os registradores A, B, C' e C″ operam como registradores de deslocamento (shift-registers),
- Paralelo, quando este sinal está com valor '1', os registradores A, B, C' e C″ são carregados paralelamente.

Esta operação e' necessária pois os valores dos coeficientes são alimentados paralelamente pelo pre'-processador, no entanto, estes dados são repassados às UAL'S serialmente.

A unidade de controle gera como sinais de saída para os multiplicadores (Figura 27 na página 55):

- Iterum, indica que um dado válido foi carregado no primeiro nível do multiplicador,
- Iterdois, indica que um dado válido foi carregado no segundo nível do multiplicador,
- Itertrês, indica que um dado válido foi carregado no terceiro nível do multiplicador.

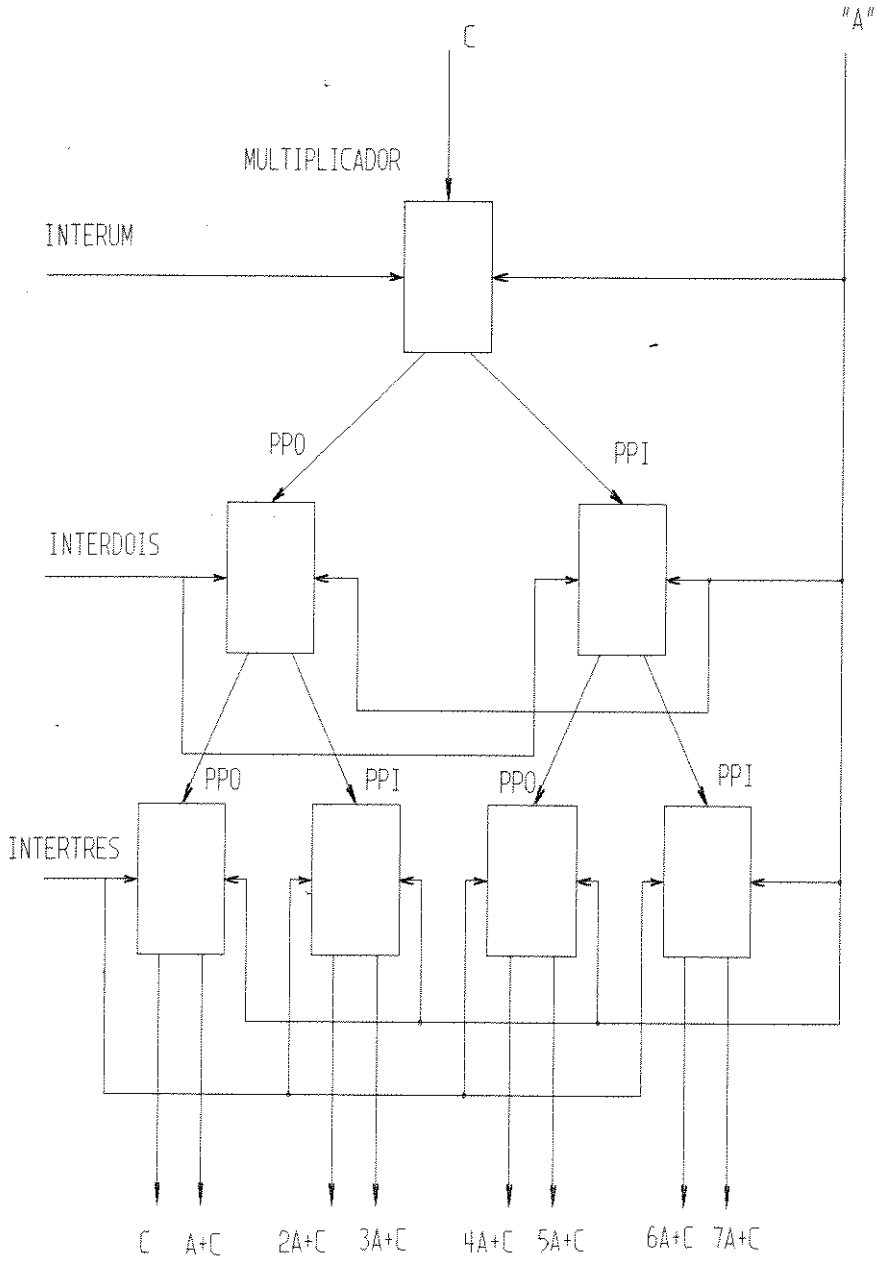


Figura 27. Esquema Geral do Multiplicador

Este sinais controlam a realimentação do carry-out (Cout) nas células de multiplicação, assegurando que o carry-in (Cin) seja zero, enquanto o dado válido do nível anterior não chega, como podemos observar no detalhe da célula de multiplicação na Figura 23 na página 48.

A unidade de controle interage com as UAL's através dos seguintes sinais de controle (Figura 24 na página 49):

- Tree,
- Go,
- Prime,
- Set e
- Timer.

Quando o sinal Tree (y-tree ou x-tree) esta' em '1', as UAL's receberão como entradas as saídas dos multiplicadores, permitindo assim que o somador realize a operação $(Ax+C')+(By+C'')$. Quando em '0', as UAL's serão alimentadas com a saída serial dos registros F e Z, através das linhas Memo1 e Memo2. A linha Memo2 será complementada permitindo a operação F-Z.

O sinal Go indica quando os dados que alimentam as UAL's são válidos. Quando este sinal não está selecionado ($GO=0$), ele desabilita todas as saídas Write Enable e bloqueia a entrada de dados no somador das UAL's.

A saída Prime da UC indica quando o primeiro bit de um dado válido e' alimentado nas UAL's. Sua função e' garantir a realimentação correta do Cout, fazendo com que 0's sejam introduzidos em Cin quando dados do multiplicador são alimentados nas UAL's e que 1's sejam introduzidos em Cin quando os dados vierem dos registros Z e F , porque a operação F-Z e' realizada em complemento de 2.

O sinal Set habilita todos os registros Enable e o sinal Timer indica quando estes registros podem mudar de estado.

A unidade de controle controla as 64 memórias de pixels através dos seguintes sinais de controle:

- Habilita Z, que permite leitura serial de Z para comparação e a transferência paralela do conteúdo de F para Z caso o sinal Write Enable esteja habilitado,
- Habilita I, que permite a transferência em paralelo do conteúdo de I para P, caso Write Enable esteja habilitado,
- Shift, caso este sinal esteja habilitado ('1') Z e F operam como registros de deslocamento; caso contrário, o registro Z recebe os dados de F paralelamente e o valor de F e' congelado,
- UAL, que controla o mux existente na entrada dos registros F e Z, e seleciona se estes registradores serão alimentados pela saídas das UAL's ou realimentados pelos próprios registradores quando em modo registrador de deslocamento.

6.0 Conclusão

O objetivo inicial deste trabalho era analisar os diferentes algoritmos para eliminação de linhas e superfícies escondidas para posterior implementação de um ou mais destes algoritmos em software.

Durante o estudo destes algoritmos, no entanto, foi constatado que o algoritmo Z-buffer poderia ser implementado através de um circuito integrado. Esta constatação foi de grande valia, pois reunia dois assuntos de grande interesse do autor ou seja, computação gráfica e arquiteturas VLSI. Foi então proposta a implementação do algoritmo usando a técnica de gate-arrays. Henry Fuchs e John Poulton haviam apresentado uma solução utilizando a técnica full-custom que resulta num tempo de desenvolvimento e custo maiores /Fuch81/.

Esta nova implementação levou-nos a propor uma série de mudanças para adaptar a arquitetura anteriormente proposta para uma técnica onde o grau de liberdade de projeto e' menor.

O resultado deste trabalho, como pode ser visto pelos resultados das simulações apresentados no Anexo, atingiu às especificações descritas no capítulo 4. O circuito realiza as operações de comparação de Z com F, avaliando qual e' o menor Z naquele pixel a partir dos coeficientes gerados pelo pre'-processador.

O circuito proposto por /Fuch81/ possuía algumas diferenças básicas com o proposto neste trabalho. O circuito de Fuchs não possuía a Unidade de Controle, o controle tinha de ser efetuado externamente; para isso era necessário um barramento de 11 bits por onde eram comandadas as operações de carga das UAL's, comparação entre os registros, etc. Com isso o circuito de Fuchs exigia uma unidade de controle externa, ou mesmo, um microprocessador com o firmware adequado. Achamos, no entanto, que este circuito poderia ser integrado ao processador de pixels fazendo com que sua interação com o meio externo se desse apenas com 4 sinais de controles, isto reduz o número de pinos do CI, que o faz requerer um encapsulamento mais barato, e também permite que um eventual controlador se ocupe de tarefas mais complexas. Outra grande diferença entre os dois circuitos reside nos registros Z, F e UAL. Fuchs utilizou uma técnica possível de ser empregada em circuitos full-custom e utilizou registradores dinâmicos. Este tipo de circuito faz com que a complexidade da estrutura seja menor que registradores estáticos, como foram usados no circuito apresentado neste trabalho, no entanto, exigem um controle mais sofisticado na operação, fazendo com que a Unidade de Controle seja mais complexa.

Como prosseguimento do trabalho aqui apresentado podemos sugerir a interligação de vários processadores de pixels de maneira a processar um número maior de pixels. Cada processador de pixels processa 64 pixels, no entanto interligados de uma maneira apropriada, cada um pode processar parte da tela. Para isso e' necessário um circuito de carga adicional nos multiplicadores de maneira a permitir que a mesma estrutura seja usada para diferentes porções da tela. Um circuito de seleção de cada chip também e' necessário, com isso o controlador do SV pode controlar qual dos processadores de pixels esta' selecionado.

Pelos resultados da simulação podemos observar a rapidez de processamento do processador de pixels, carregados os coeficientes, o circuito e' capaz de apresentar o resultado da comparação de F com Z em apenas 2.3 microsegundos. Esta velocidade

de processamento e' devida a estrutura e também devida a tecnologia que foi usada neste chip. A tecnologia Toshiba 1.5 mícrons tem um atraso de porta da ordem de 1 nanosegundo, permitindo o uso de frequência de ate' 100 Megahertz. Este chip, caso fosse fundido, seria um gate-array com complexidade equivalente a 5 microprocessadores 8085, ou seja 25.0000 gates equivalentes. Ele possui um barramento de dados (D0-7) de 8 bits, um barramento de controle de 4 bits, duas entradas de clock e um barramento de saída (P0-7) de oito bits. Foi utilizada a técnica de dois clocks que não se sobrepõem (non-overlapping).

O circuito foi simulado exaustivamente e os casos de simulação foram conclusivos quanto à funcionalidade do mesmo. Estes simuladores apresentam uma alta confiabilidade inclusive incorporando atrasos que seriam encontrados nas linhas após o projeto fisico.

A próxima etapa de projeto de um circuito integrado e' o projeto fisico e posterior fundição. Estes dois últimos passos, no entanto, não foram realizados pois possuem custos elevados e apresentam pouco interesse acadêmico. Hoje um simples passo para fundição deste tipo de circuito pode chegar a 20.000 dólares. Além do mais, o projeto fisico e' realizado na própria foundry através de roteadores automáticos com pouca ou nenhuma intervenção humana.

O autor espera, com este trabalho, ter contribuído para a melhoria do entendimento de arquitetura VLSI aplicada a soluções em computação gráfica, onde se requeira soluções de alta performance e alta compactação.

7.0 Bibliografia

- /App67/-Appel, A.: "The Notion of Quantitative Invisibility and the Machine Rendering of Solids", Proc. ACM National Conference, 1967, pp 387-393.
- /Bouk69/-Bouknight, W. J.: "An Improved Procedure for Generation of Half-Tone Computer Graphics Representations", University of Illinois, 1969.
- /Catm74/-Catmull, E.: "A Subdivision Algorithm for Computer Display of Curved Surfaces", Ph.D. Thesis, Univeristy of Utah, 1974.
- /Fuch81/-Fuchs, H. e Poulton, J.: "Pixel-Planes: A VLSI-Oriented Design for a Raster Graphics Engine", 1981, VLSI Design Magazine, pp 20-28.
- /IEEE77/-"Semiconductor face the 80's", IEEE Spectrum October 1977, pp 42-48.
- /Joy88/-Joy, K. I. e Grant, C. W.: "SIGGRAPH '88 Tutorial no.9 Image Synthesis", 1988.
- /Lou67/-Loutrel, P.: "A Solution to the Hidden-line Problem for Computer Drawn Polyedra", New York University Technical Report 400-167, Sept. 1967.
- /Newe72/-Newell, M. E.; Newell, R. G. and Sancha, T. L.: "A New Approach to the Shaded Picture Problem", Proc. ACM National Conf., 1972.
- /Pini86/-Magalhães,Léo Pini : "Computação Gráfica", 1a. edição, Editora da Unicamp, 1986.
- /Read85/-Read, John : "Gate Arrays", McGraw Hill, 1985.
- /Rob63/-Roberts, L. G.: "Machine Perception of Three-Dimensional Solids", MIT Lincoln Laboratory, 1963.
- /Romn70/-Romney, G. W.: "Computer Assisted Assembly and Rendering of Solids", Univ. of Utah, TR 4-20, 1970.
- /Spro79/-Sproull, R. F. e Newman, W. M.: "Principles of Interactive Computer Graphics",1979, 2a. edição, McGraw Hill.
- /Suth74/-Sutherland, I. E.; Sproull, R. F. e Schumacker R. A.: "A Characterization of Ten Hidden-surface Algorithms", Computing Surveys, vol. 6, no. 1, March 1974.
- /Warn69/-Warnock, J. E. : "A Hidden-Surface Algorithm for Computer-Generated Halftone Pictures", University of Utah Technical Report 4-15, 1969.
- /Watk70/-Watkins, G. S.: "A Real Time Visible Surface Algorithm", University of Utah, UTECH CSC 70-101, 1970.
- /Wili67/-Wilie C.; Romney, G.; Evans, D. C. and Erdahl, A.: "Halftone Perspective Drawings by Computer", Proc AFIPS JCC, vol. 31, 1967.

8.0 Anexo

8.1 Sistema EDS

O Engineering Design System da IBM e' o sistema responsável pelo projeto de sistemas, subsistemas e módulos da IBM. Ele vem sendo utilizado na companhia desde 1965, quando operava em cpu's 360 e 370. Este sistema foi um dos primeiros a ser utilizado para se projetar outros sistemas. Desde 1965, o EDS vem sofrendo uma série de atualizações e inclusões visando novas tecnologias; com ele e' possivel projetar, utilizando um banco de dados único, desde os circuitos integrados de um novo sistema ate' simular funcionalmente uma CPU 3090.

O EDS e' um sistema hierárquico. Os módulos, que são projetados utilizando o estilo gate-array, standard cells ou full-custom, podem ser simulados a nível de subsistema (nível de cartão) e a nível de sistema completo, antes mesmo de serem fabricados.

As etapas de projeto no EDS estão mostradas na Figura 28 na página 61.

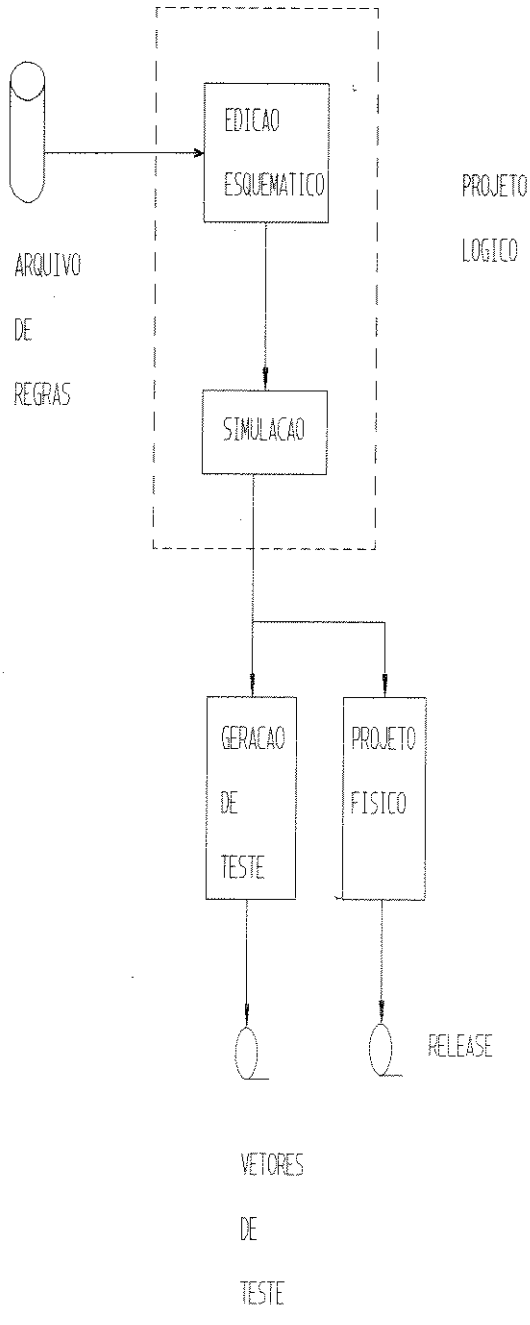


Figura 28. Etapas de projeto no EDS

Na edição de esquemático o circuito projetado e' introduzido no sistema utilizando um editor de esquemático chamado TILES (Toronto Interactive Logic Entry System). O projeto e' descrito em termos das portas lógicas existentes na tecnologia, tais como

Nand's, Or's, etc. Após a introdução do circuitos, o TILES gera um arquivo chamado de BDLS (Basic Description Language for Structure).

A BDLS e' uma linguagem de descrição lógica. Como vimos no capítulo 3, ela contém informações tais como a interligação das várias portas, características de cada porta e modelos, que serão usados nas fases seguintes do projeto: na simulação, na geração automática de testes e no projeto fisico.

Na Figura 29 na página 63 esta' apresentada uma descrição do circuito da UAL em BDLS:

```
SHEET ALU01,LOC = 1A-A1A1A30A30AA01,
        T1 = UAL0;

ALU01AA(10/ALU01AA10) = = N,IVVH.A.,N(A0/TREE1BA10)::PP = 15B,ENG = (A0/A,
  10/Z),EOB = (10/'&');
ALU01AH(10/ALU01AH10) = = N,IVVH.A.,N(A0/PRIMEBA10)::PP = 15W,ENG = (A0/A,
  10/Z),EOB = (10/'&');
ALU01BA(10/ALU01BA10) = = AI,ND2L.A.,N(A0/ALU01AA10,A1/REG01DB30)::PP = 25B,
  ENG = (A0/A,A1/B,10/Z),EOB = (10/'&');
ALU01BB(10/ALU01BB10) = = AI,ND2L.A.,N(A0/TREE1BA10,A1/YMULTEG10)::PP = 25E,
  ENG = (A0/A,A1/B,10/Z),EOB = (10/'&');
ALU01BD(10/ALU01BD10) = = AI,ND2L.A.,N(A0/ALU01AA10,A1/REG01DF30)::PP = 25K,
  ENG = (A0/A,A1/B,10/Z),EOB = (10/'&');
ALU01BE(10/ALU01BE10) = = AI,ND2L.A.,N(A0/TREE1BA10,A1/XMULTEG10)::PP = 25N,
  ENG = (A0/A,A1/B,10/Z),EOB = (10/'&');
ALU01BG(10/ALU01BG10) = = AI,ND2L.A.,N(A0/ALU01AA10,A1/PRIMEBG10)::PP = 25T,
  ENG = (A0/A,A1/B,10/Z),EOB = (10/'&');
ALU01BH(10/ALU01BH10) = = AI,ND2L.A.,N(A0/ALU01GB10,A1/ALU01AH10)::PP = 25W,
  ENG = (A0/A,A1/B,10/Z),EOB = (10/'&');
ALU01CA(10/ALU01CA10) = = AI,ND2L.A.,N(A0/ALU01BA10,A1/ALU01BB10)::PP = 35B,
  ENG = (A0/A,A1/B,10/Z),EOB = (10/'&');
ALU01CD(10/ALU01CD10) = = AI,ND2L.A.,N(A0/ALU01BD10,A1/ALU01BE10)::PP = 35K,
  ENG = (A0/A,A1/B,10/Z),EOB = (10/'&');
ALU01CG(10/ALU01CG10) = = AI,ND2L.A.,N(A0/ALU01BG10,A1/ALU01BH10)::PP = 35T,
  ENG = (A0/A,A1/B,10/Z),EOB = (10/'&');
ALU01DA(10/ALU01DA10) = = XOR,EOPL.A.,N(A0/ALU01CA10,B0/ALU01DE10)::PP = 45B,
  ENG = (A0/A,B0/B,10/Z);
ALU01DB(10/ALU01DB10) = = AI,ND2L.A.,N(A0/ALU01CA10,A1/ALU01DE10)::PP = 45E,
  ENG = (A0/A,A1/B,10/Z),EOB = (10/'&');
ALU01DC(10/ALU01DC10) = = AI,ND2L.A.,N(A0/ALU01EA10,A1/ALU01FC30)::PP = 45H,
  ENG = (A0/A,A1/B,10/Z),EOB = (10/'&');
ALU01DE(10/ALU01DE10) = = XOR,EOPL.A.,N(A0/ALU01CD10,B0/ALU01AA10)::PP = 45N,
  ENG = (A0/A,B0/B,10/Z);
ALU01EA(10/ALU01EA10) = = XOR,EOPL.A.,N(A0/ALU01DA10,B0/ALU01CG10)::PP = 55B,
  ENG = (A0/A,B0/B,10/Z);
ALU01EB(10/ALU01EB10) = = AI,ND2L.A.,N(A0/ALU01DA10,A1/ALU01CG10)::PP = 55E,
  ENG = (A0/A,A1/B,10/Z),EOB = (10/'&');
ALU01EC(10/ALU01EC10) = = AI,ND2L.A.,N(A0/SETBUBA10,A1/ALU01DC10)::PP = 55H,
  ENG = (A0/A,A1/B,10/Z),EOB = (10/'&');
ALU01FB(10/ALU01FB10) = = AI,ND2L.A.,N(A0/ALU01EB10,A1/ALU01DB10)::PP = 65E,
  ENG = (A0/A,A1/B,10/Z),EOB = (10/'&');
ALU01FC(30/ALU01FC30) = = SRL,SRBH.A.,N(A0/ALU01FD10,B0/B0BUFBA10,
  C0/C0BUFBA10,D0/ALU01EC10)::PP = 65H,ENG = (A0/ACLOCK,B0/BCLOCK,
  C0/C1CLOCK,D0/DATA,30/L2PL);
ALU01FD(10/ALU01FD10) = = TIE,CLIP.A.,N::PP = 65K,EOB = (10/'&');
ALU01GB(10/ALU01GB10) = = DFF,FD1L.A.,N(D0/ALU01FB10,G0/PH1BUBA10)::PP = 75E,
  ENG = (D0/D,G0/CP,10/Q);
ALU01GC(10/ALU01GC10) = = AND,AN2H.A.,N(A0/ALU01FC30,A1/GOBUFBA10)::PP = 75H,
  ENG = (A0/A,A1/B,10/Z);
```

Figura 29. UAL descrita em BDLS

Na edição do esquemático e' necessária a definição a priori da tecnologia que será empregada, para isto um biblioteca de células e' utilizada. Cada tecnologia possui sua biblioteca que contém os modelos lógicos, atrasos de portas e demais informações que descrevemos anteriormente. Uma célula de um inversor INV, por exemplo, tem no seu modelo lógico a descrição da sua função:

```
If in = 0 then out = 1;
```

else out = 0;

onde in e' a entrada e out a saída do inversor.

Este modelo e' utilizado pelo simulador para verificação do funcionamento do circuito.

A BDLS não e' uma forma natural de entendimento para o projetista, para facilitar a compreensão e a documentação dos projetos são utilizados diagramas lógicos chamados de ALD's. Estes diagramas são apresentados no anexo deste trabalho. Eles apresentam a arquitetura do circuito e suas interligações dentro e entre as páginas. Está e' a representação de diagramas lógicos que e' usada internamente na IBM.

Cada quadrado representado nas páginas significa uma porta lógica e sua função pode ser encontrada através da leitura da primeira linha escrita dentro dele; por exemplo, um And e' representado pela letra A, um Or por OR, um flip-flop por FF, etc. Caso a saída de uma porta seja invertida, por exemplo, um Nand, o quadrado estara' representado pela letra A e sua saída terá um pequeno triângulo indicando que este representa um Nand. Portanto, caso tenhamos um inversor, este terá a letra N internamente e um pequeno triângulo na saída indicando que sua saída e' inversora.

Através da leitura do sinal e' possível saber em que página o sinal está sendo gerado bem como para onde o sinal está sendo transmitido. O nome de cada página se encontra no canto inferior direito. Todas as entradas de cada página estão no lado esquerdo da mesma e as saídas no lado direito. Sinais de entrada e saída são colocados no lado esquerdo.

Vejamos o exemplo da página ALU001: o sinal WRITE DATA no canto superior direito da página está sendo transmitido para a página REG01. Nesta página podemos ver que este sinal foi gerado na página ALU01 pelo bloco EA saída 1(ou 10).

8.2 Análise Geral da Simulação

Toda a simulação do processador de pixels foi realizada utilizando um dos simuladores disponíveis no EDS: o simulador Aussim. Este simulador e' interativo, permitindo que o projetista controle os sinais de entrada e observe o funcionamento detalhado do circuito. A qualquer tempo e' possível observar qualquer sinal existente externa ou internamente no chip, para isto uma tela com os mesmos e' mostrada como se fosse uma tela de um analisador lógico, ou seja, os sinais são mostrados como forma de onda, tendo o tempo como parâmetro no eixo x.

A idéia básica da simulação e' estimular todas as partes envolvidas na operação do processador de pixels, através de suas 12 entradas, a saber:

- barramento de dados: D0-D7,
- endereço,
- dado,
- reset,
- edge.

E, observar os resultados obtidos nos registradores Z, F, I e P.

O esquema geral da simulação e':

1. carregar os valores de A, B, C' e C'' de maneira a obtermos os seguintes casos:
   a. $F(x,y) < 0$

Neste caso, nenhuma operação sera' realizada nas UAL's, portanto o valor de Z deve ser mantido constante e nenhuma mudança ocorrera' nos registros I e P.

b. $F(x,y) = 0$

c. $F(x,y) > 0$

Nos dois últimos casos, o pixel faz parte do polígono que esta' sendo processado portanto uma comparação entre os valores de F e Z se faz necessária. Para estes casos devemos ter dois conjuntos de valores dos coeficientes:

1) A, B, C' e C'' tais que $F >= Z$; neste caso o valor do registrador Z permanecera' inalterado.

2) A, B, C' e C'' tais que $F < Z$, o valor de Z sera' substituído pelo valor de F.

2. ler os valores de P através do barramento P0-7 e verificar o se este registradores foram devidamente carregados com o resultado da comparação quando esta ocorrer.

A seguir apresentamos as formas de onda obtidas na simulação através da saída do Aussim, cada operação realizada com o processador de pixels esta' comentada.

9.0 Resultado das Simulações

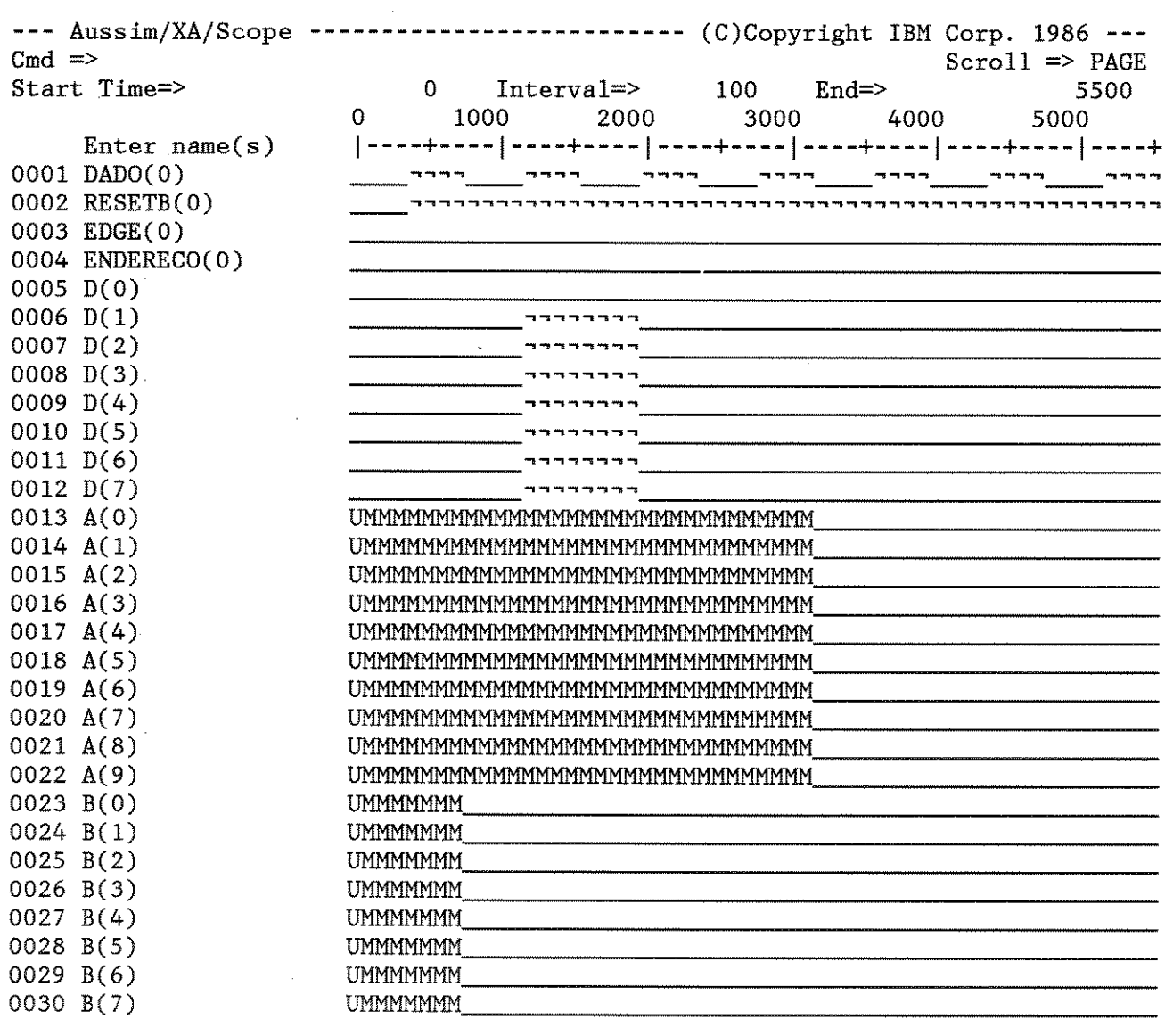
INICIALIZACAO DOS REGISTROS Z (DE 0 A 8000NS)

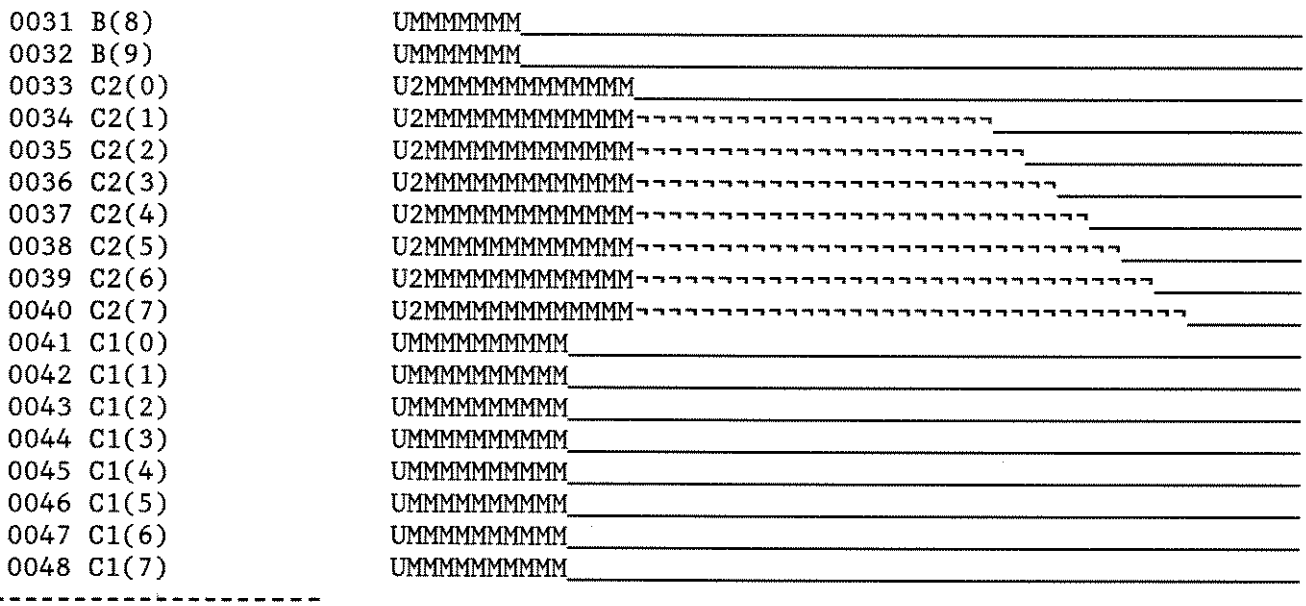
De 0 a 400NS, o processador de pixels e' resetado (RESETB=0).

Inicialmente iremos carregar todos os valores de Z com o valor maximo positivo em complemento de dois, ou seja, 7F. Para isto carregamos os valores de B, C2, C1 e A no intervalo de 400 a 3600NS com os seguintes valores B=00, C2=7F, C1=00 e A=00 .

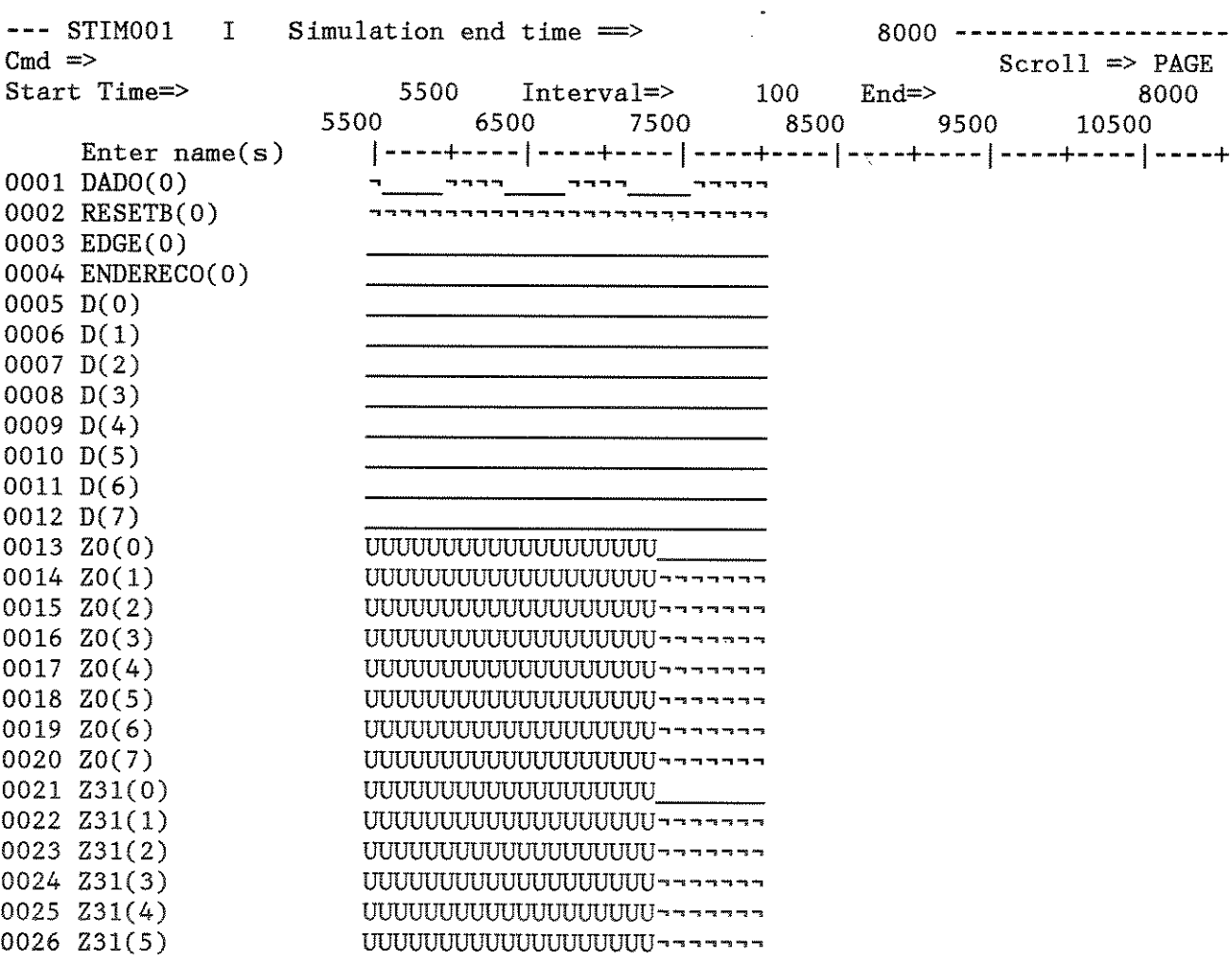
A partir de T=3600, o circuito passa a calcular F(X,Y) para os 64 pixels terminando em T=7400Ns. Pelos numeros carregados, F(X,Y)=7F para todos os pixels. Em T=7500Ns este valor e' finalmente colocado nos registros Z.

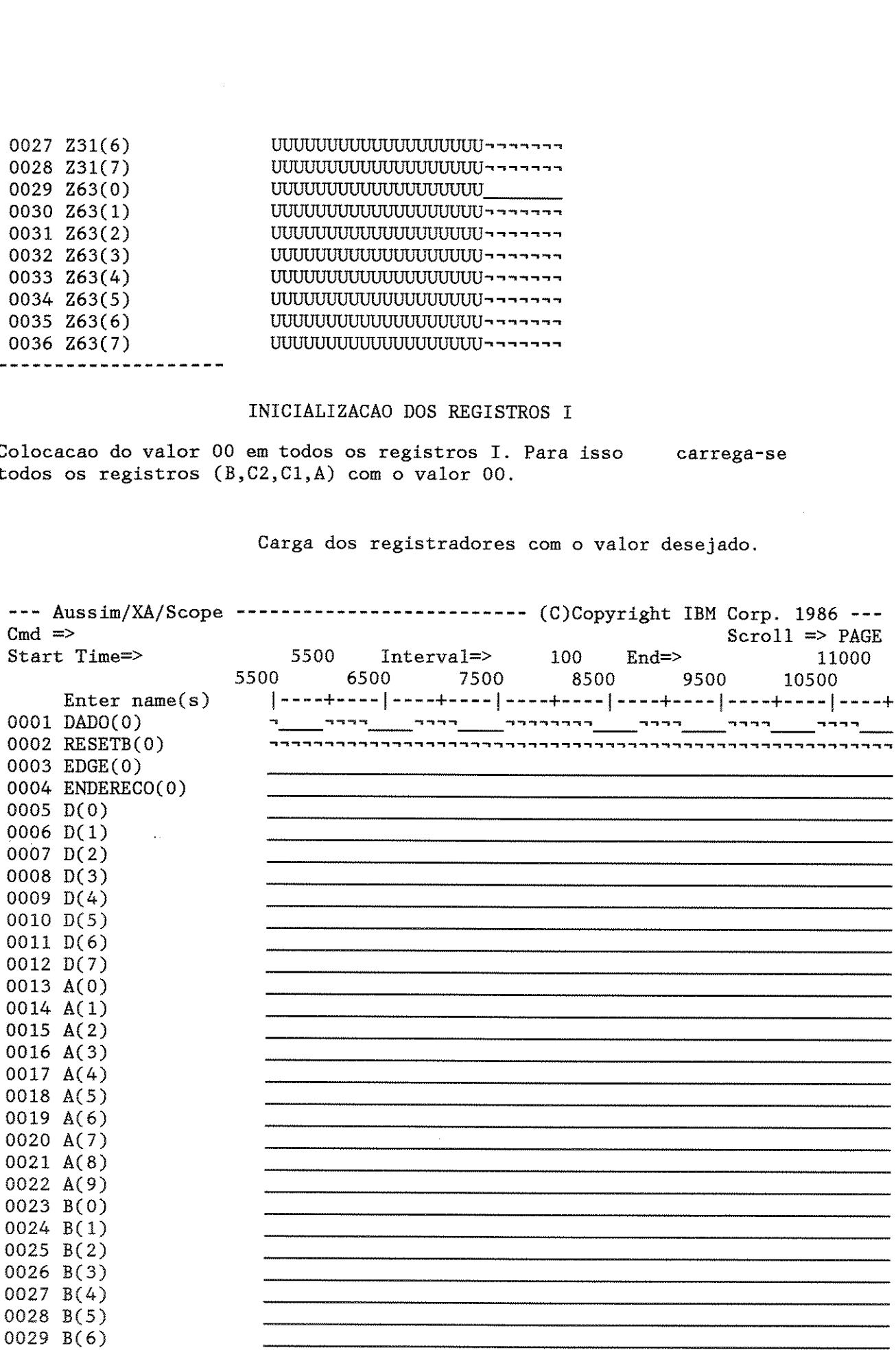
Carregamento dos registros B,C2,C1,A





Transferencia para os registros Z

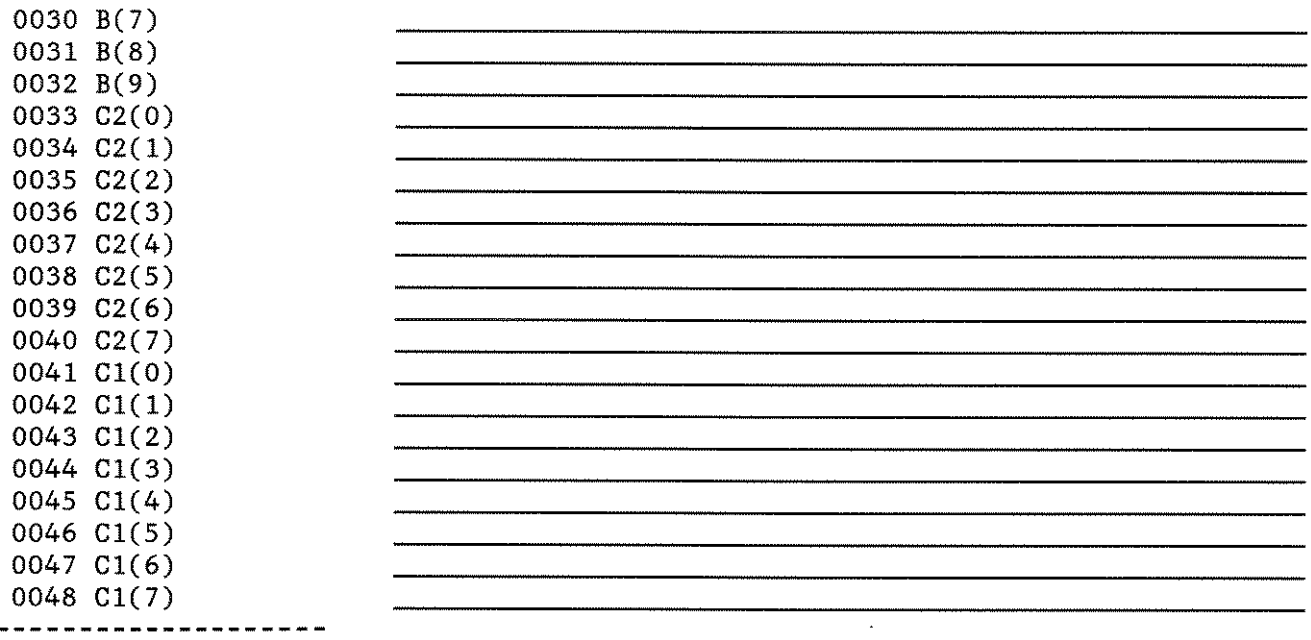




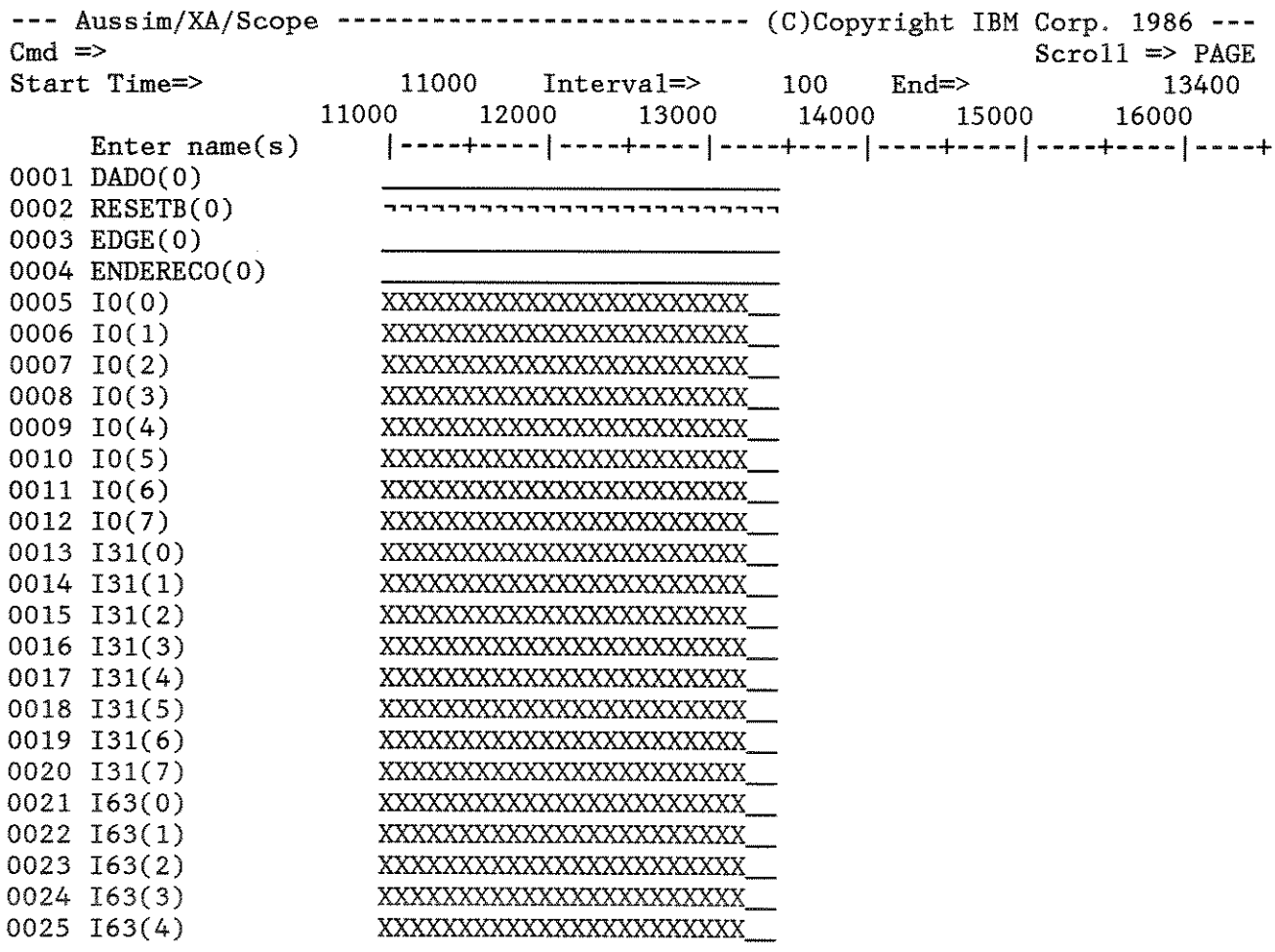
INICIALIZACAO DOS REGISTROS I

Colocacao do valor 00 em todos os registros I. Para isso carrega-se todos os registros (B,C2,C1,A) com o valor 00.

Carga dos registradores com o valor desejado.



Transfer{ncia dos valores desejados para I.

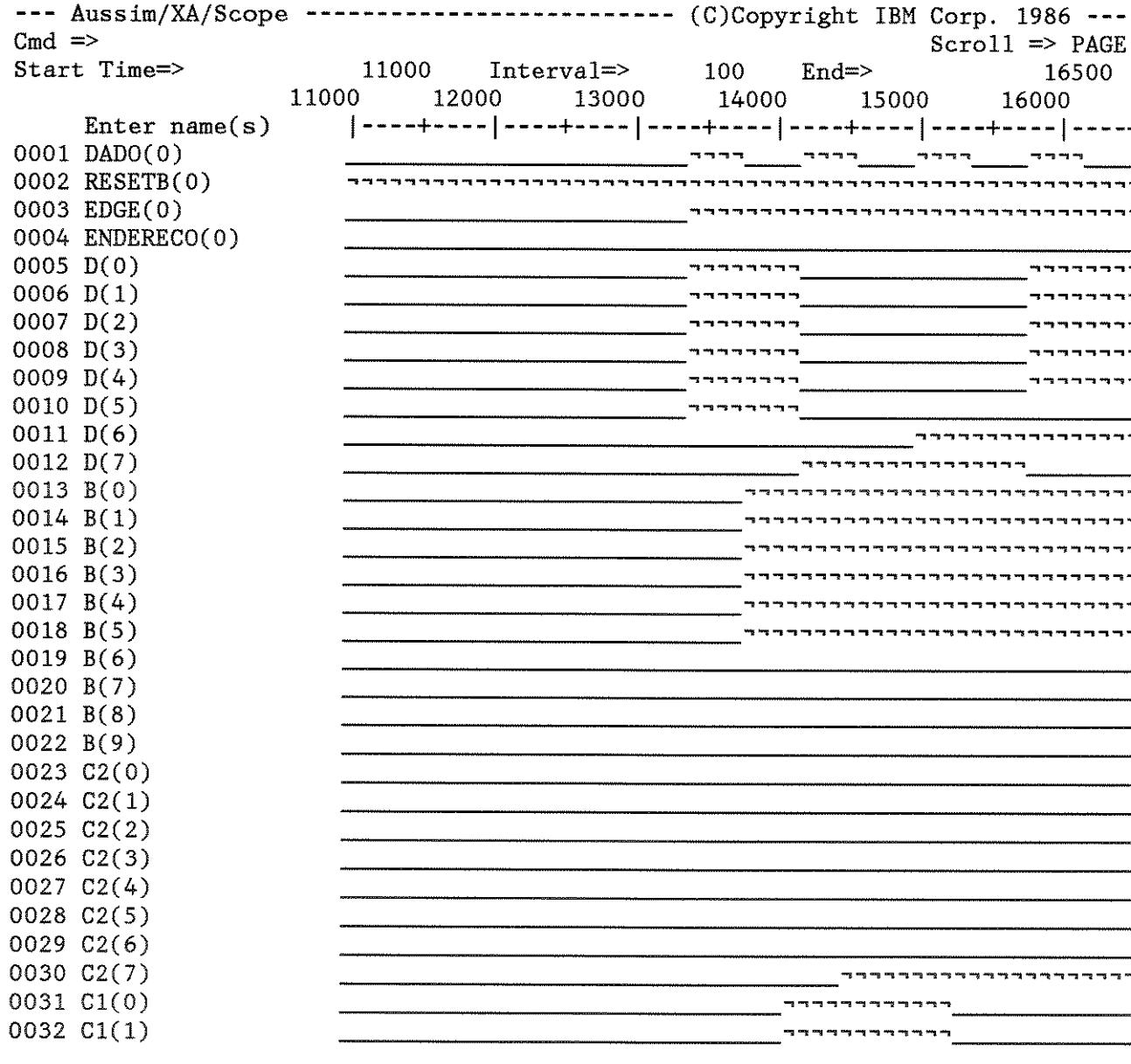


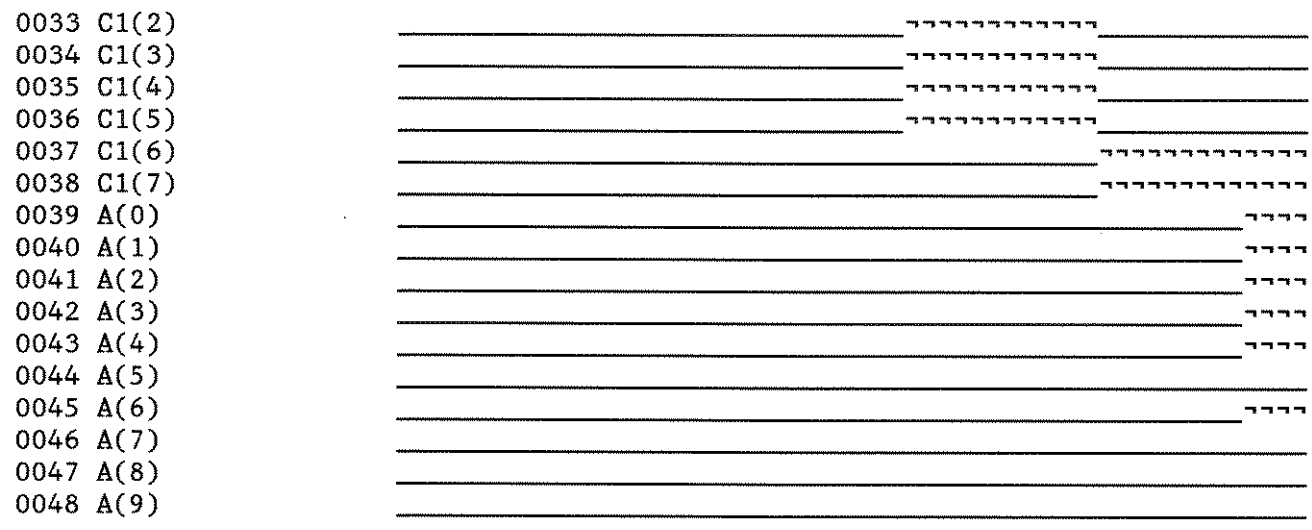
```
 0026 I63(5)              XXXXXXXXXXXXXXXXXXXXXXX__
 0027 I63(6)              XXXXXXXXXXXXXXXXXXXXXXX__
 0028 I63(7)              XXXXXXXXXXXXXXXXXXXXXXX__
--------------------
```

VERIFICACAO SE O PIXEL ESTA' NO INTERIOR DE UM POLIGONO

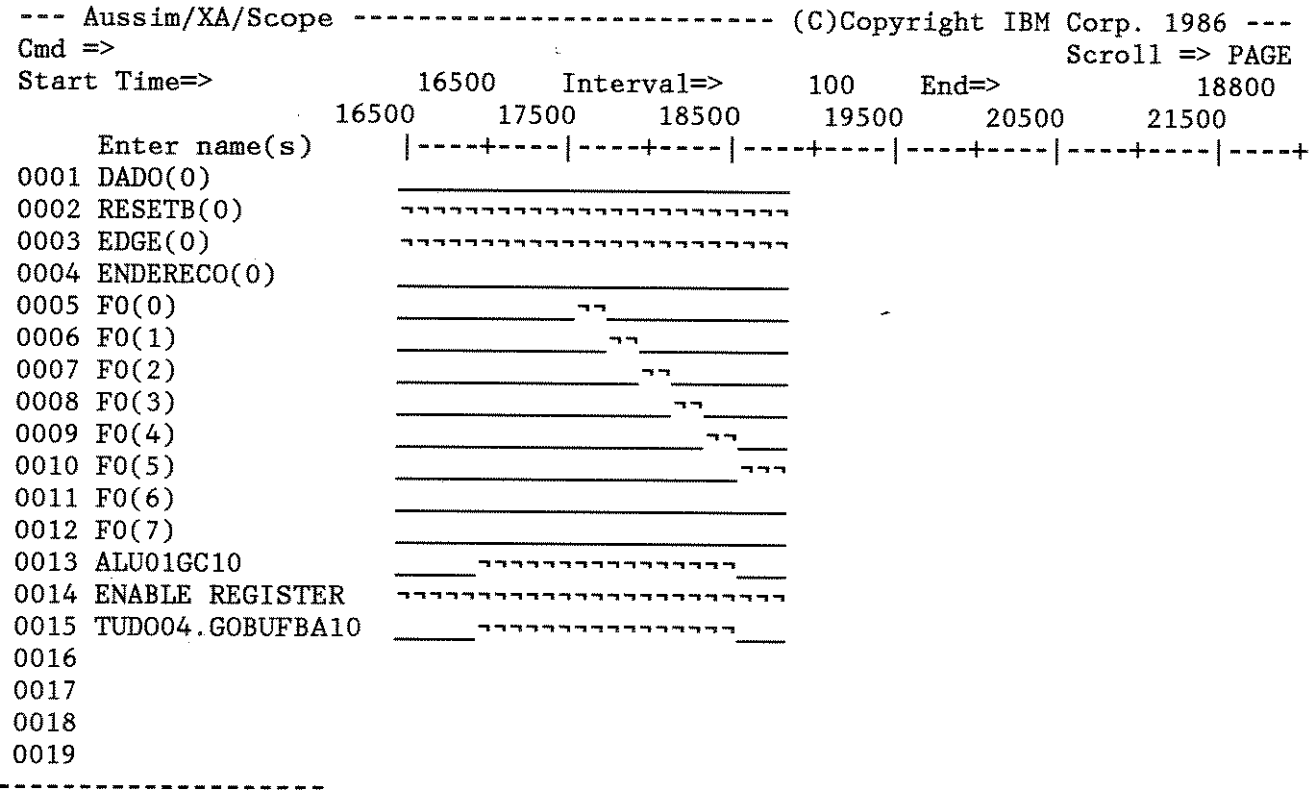
Faz- se EDGE=1 para indicar a operacao. Coloca-se os coeficientes A, B, C2 e C1 definindo uma aresta do poligono, calcula-se F(X,Y). Caso F(X,Y)<0, o pixel esta' no EXTERIOR do poligono e desabilita o enable register. Caso F(x,Y)>=0, o pixel esta' no INTERIOR do poligono e o Enable Register permanece no estado anterior. Com seguintes valores carregados nos coeficientes: B=FC, C2=03, C1=01 e A=FA. resulta em F(0,0)=04 (pixel 0), F(7,3)=CE (pixel 31) e F(7,7)=BE (pixel 63). Com isso, F0>0, F31<0, F63<0. O enable register mantera-se em um Com isso o valor da funcao nos pixels 0, 31 e 63 so respectivamente positivo, negativo e negativo.

Carga dos valores de A,B,C2,C1

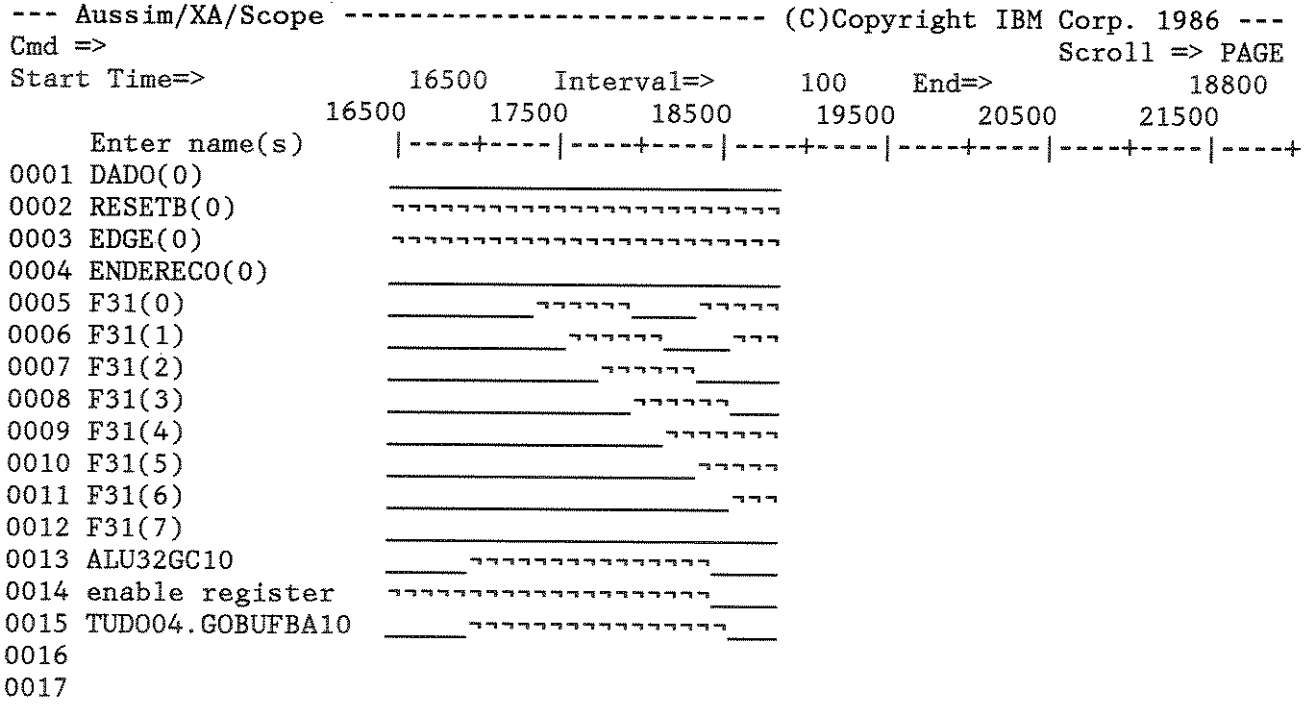



0033 C1(2)
0034 C1(3)
0035 C1(4)
0036 C1(5)
0037 C1(6)
0038 C1(7)
0039 A(0)
0040 A(1)
0041 A(2)
0042 A(3)
0043 A(4)
0044 A(5)
0045 A(6)
0046 A(7)
0047 A(8)
0048 A(9)

Resultado de F(0,0) e Sada do Enable Register



Resultado de F(7,3)



```
 0018
 0019
-------------------
```

Resultado de F(7,7)

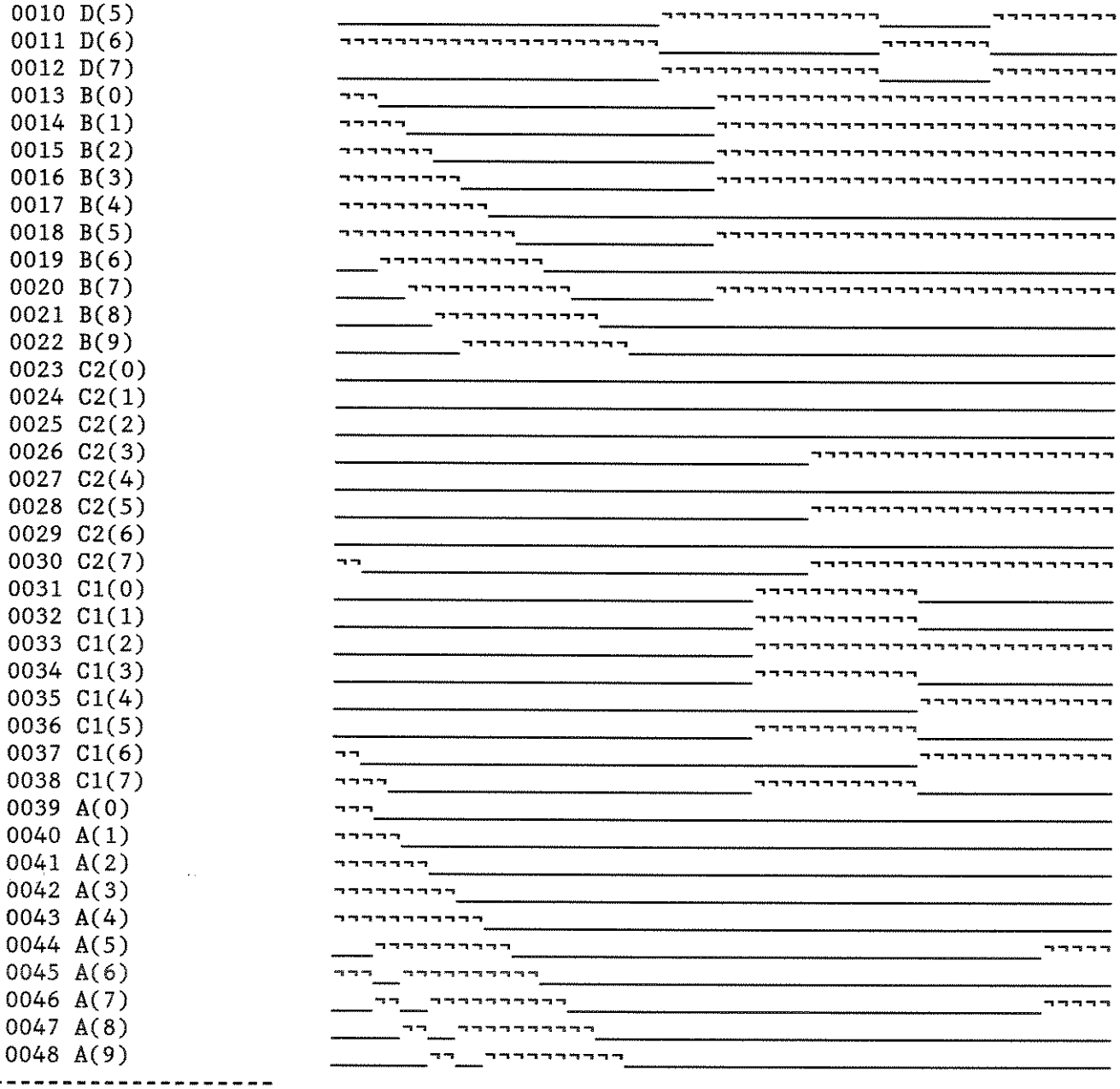
```
--- Aussim/XA/Scope ------------------------- (C)Copyright IBM Corp. 1986 ---
Cmd =>                                                          Scroll => PAGE
Start Time=>             16500    Interval=>       100    End=>          18800
                   16500     17500     18500     19500     20500     21500
     Enter name(s)     |----+----|----+----|----+----|----+----|----+----|----+
0001 DADO(0)          ________________________
0002 RESETB(0)        ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬
0003 EDGE(0)          ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬
0004 ENDERECO(0)      ________________________
0005 F63(0)           _________¬¬¬¬¬¬¬¬¬¬__¬¬¬
0006 F63(1)           ___________¬¬¬¬¬¬¬¬¬¬___
0007 F63(2)           _____________¬¬¬¬¬¬¬¬¬¬¬
0008 F63(3)           _______________¬¬¬¬¬¬¬¬¬
0009 F63(4)           _________________¬¬¬¬¬¬¬
0010 F63(5)           ___________________¬¬¬¬¬
0011 F63(6)           _____________________¬¬¬
0012 F63(7)           ________________________
0013 ALU64GC10        _____¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬____
0014 enable register  ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬____
0015 TUDO04.GOBUFBA10 _____¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬___
0016
0017
0018
0019
-------------------
```

.ce CALCULO DA PROFUNDIDADE Z

Coloca-se EDGE em zero para sinalizar a operacao de calculo de Z . Carregou-se z = F(x,y) com os seguintes valores A=05, B=F5, C2=15, C1=2A. Apenas F(0,0) sera' calculado, pois os enable registers de F31 e F63 estao desabilitados.

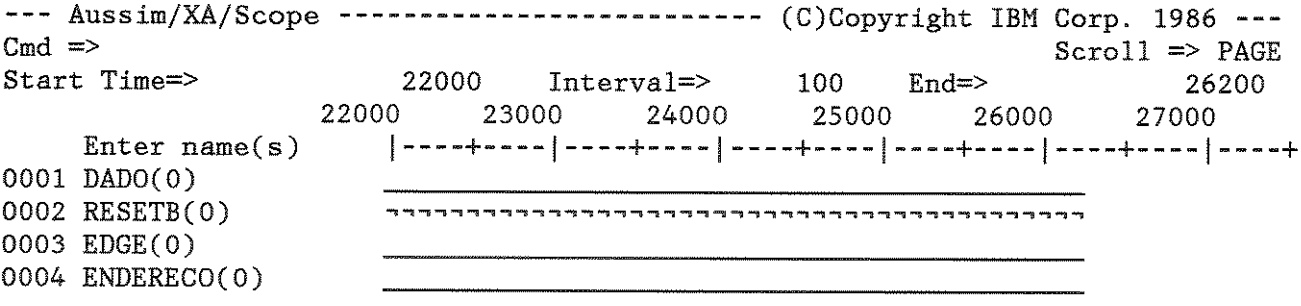
Carga de A,B,C2,C1

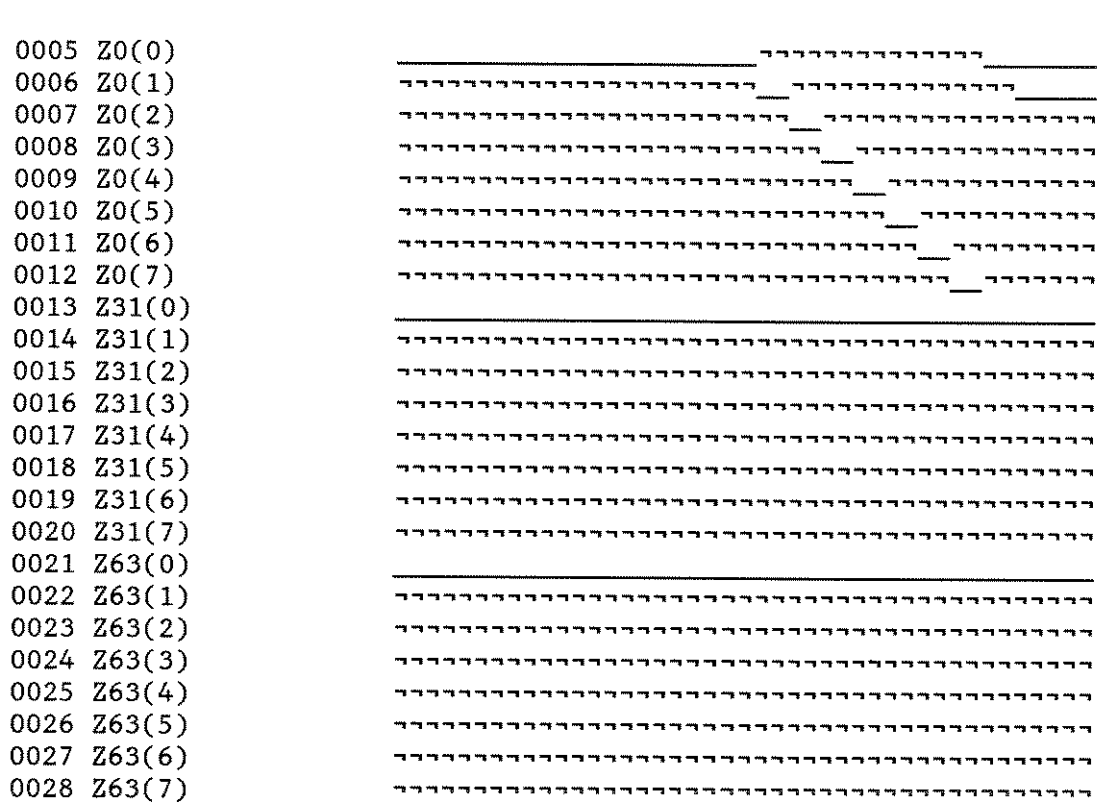
```
--- Aussim/XA/Scope ------------------------- (C)Copyright IBM Corp. 1986 ---
Cmd =>                                                          Scroll => PAGE
Start Time=>             16500    Interval=>       100    End=>          22000
                   16500     17500     18500     19500     20500     21500
     Enter name(s)     |----+----|----+----|----+----|----+----|----+----|----+
0001 DADO(0)          _______________________¬¬¬¬____¬¬¬¬____¬¬¬¬____¬¬¬¬_____
0002 RESETB(0)        ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬
0003 EDGE(0)          ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬________________________________
0004 ENDERECO(0)      _______________________________________________________
0005 D(0)             ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬________________________
0006 D(1)             ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬________________________
0007 D(2)             ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬________¬¬¬¬¬¬¬¬________
0008 D(3)             ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬________________
0009 D(4)             ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬________________¬¬¬¬¬¬¬¬________
```



Troca dos Valores do Registro Z

penas Z0 sera' trocado, visto que com os valores carregados, F(0,0)=3F
menor que o valor 7F armazenado anteriormente no registro Z. F(7,3)
F(7,7) sao calculados pois o enable register destas clulas estao
m zero. Portanto, Z31 e Z63 nao tem seus valores alterados.

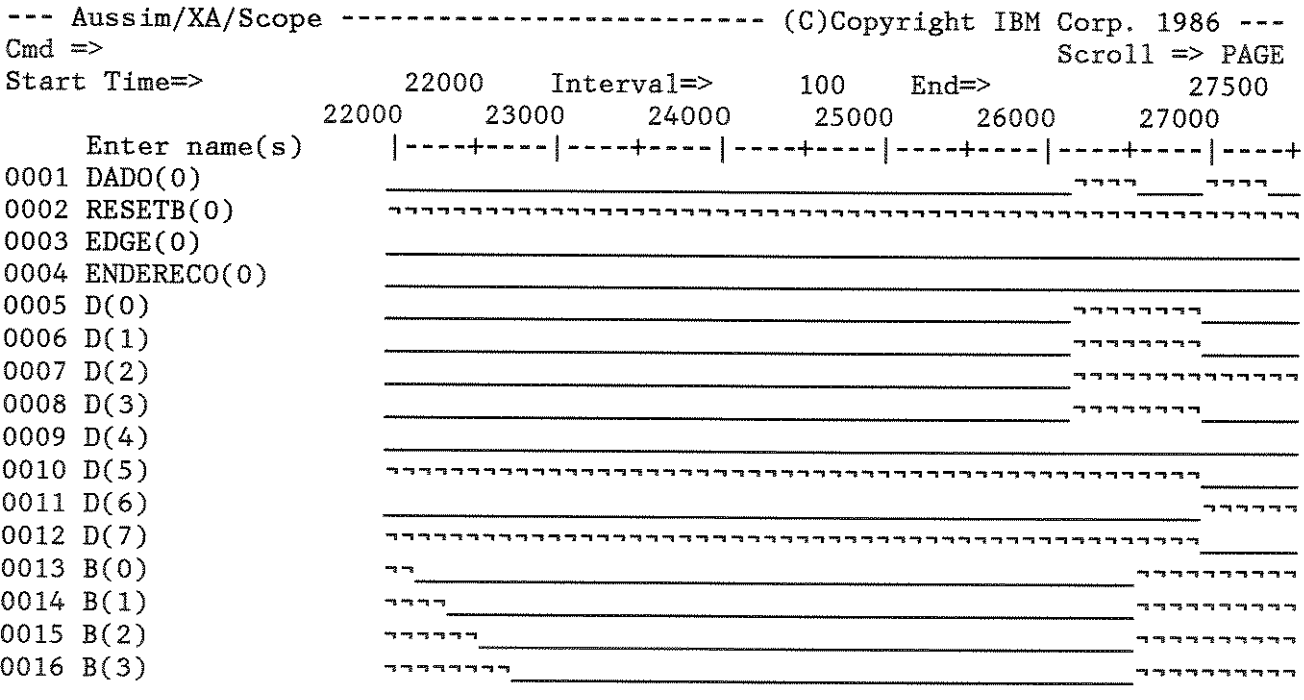


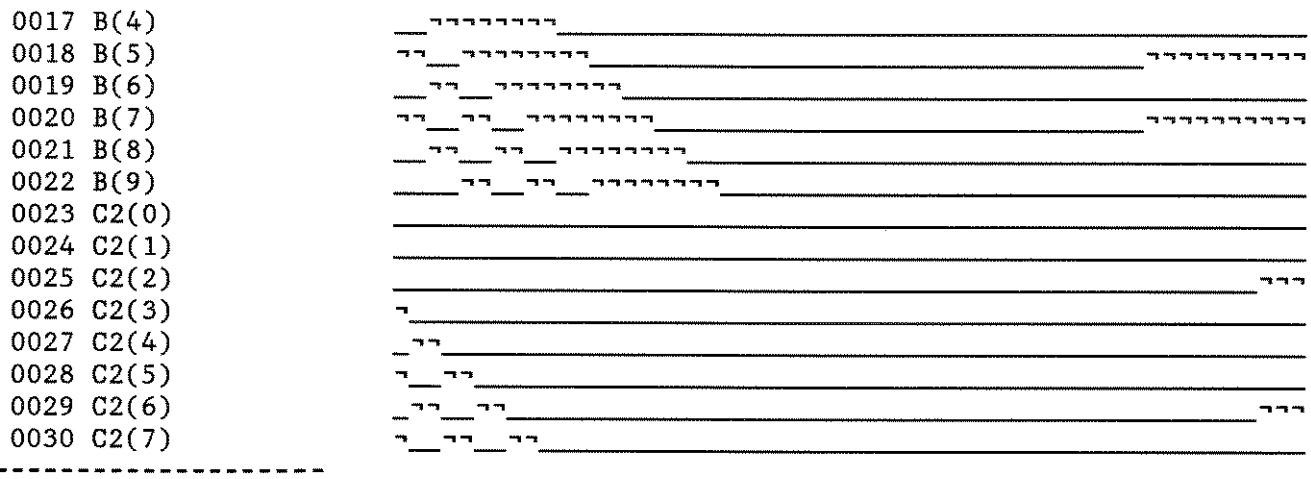


CALCULO DOS VALORES DE I

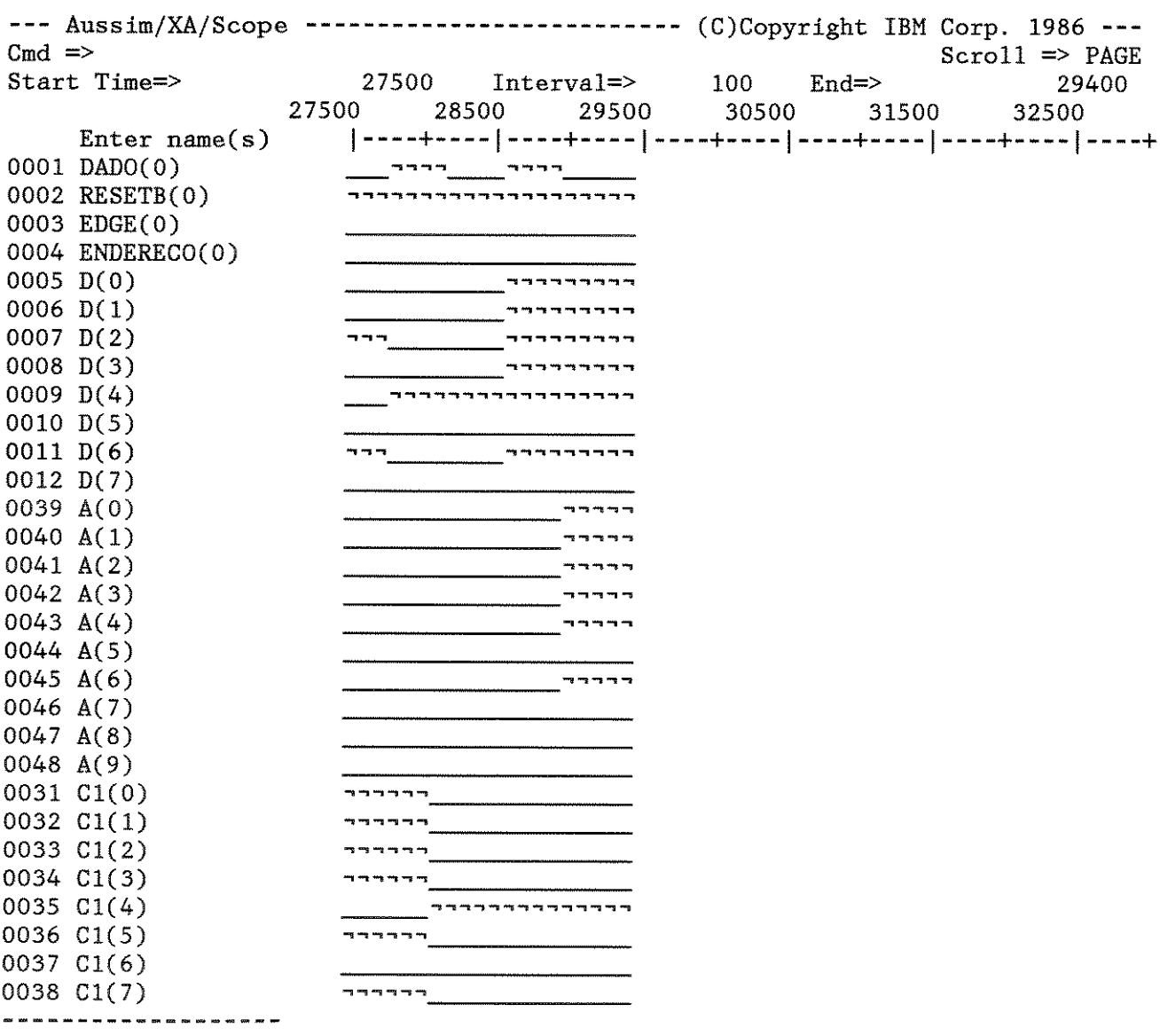
Apos um calculo de Z, os proximos valores de A, B, C2 e C1 serao interpretados como valores que definirao a intensidade no pixel. Das celulas que sao mostradas na simulacao, apenas I0 tera' seu valor alterado, pois apenas esta celula permanece com o enable register ativo. Escolheu- se A=FA, B=F5, C2=22 e C1=08 que resulta em I0=2A.

Carga dos valores de B,C2.

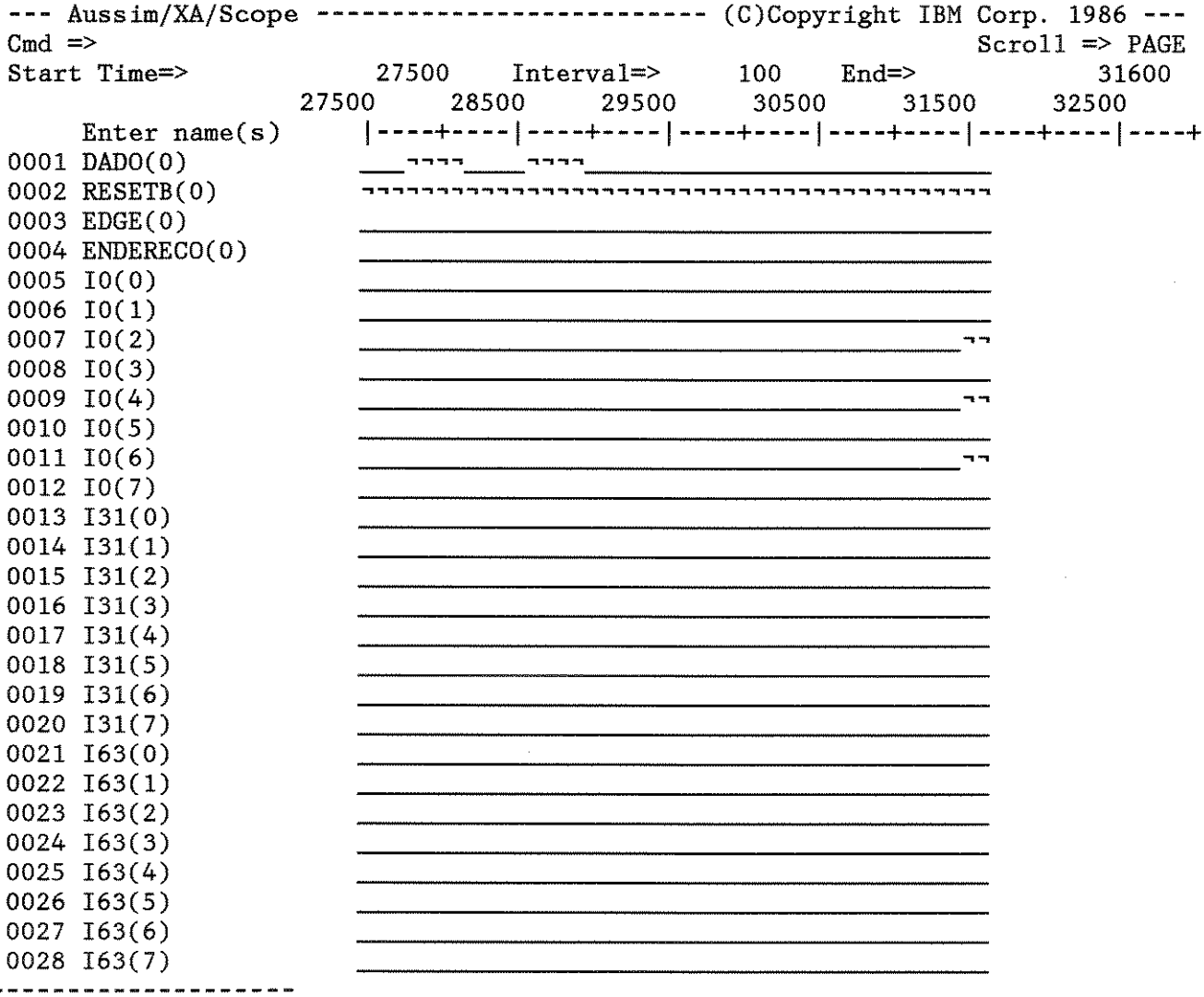




Carregamento de A e C1.

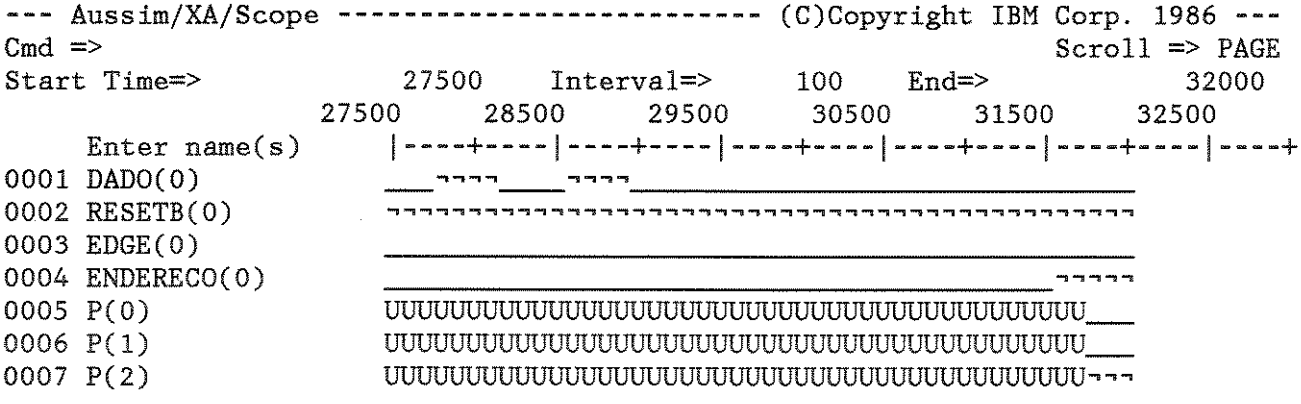


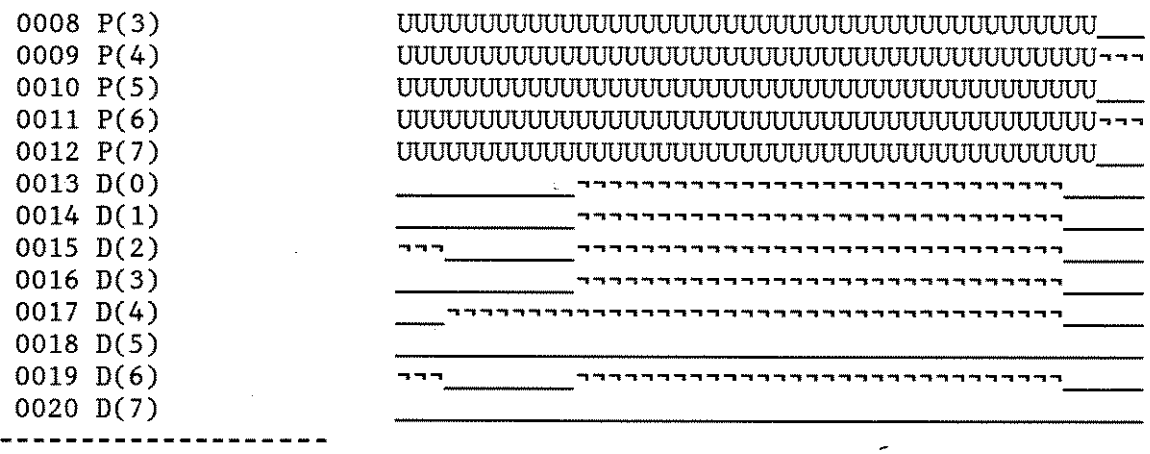
Saidas dos registros I apos a transferencia.
Como I31 e I63 estavam desabilitados, permanecem com o valor 00.

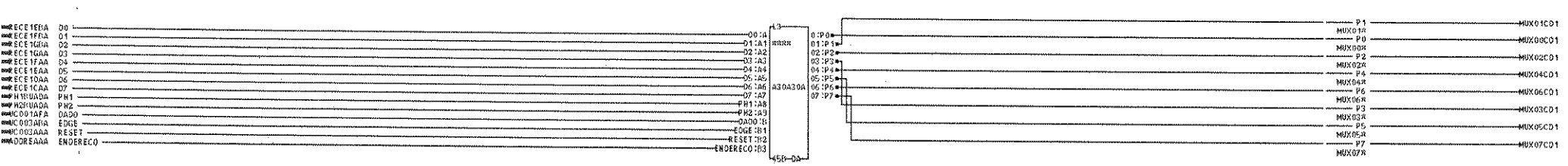


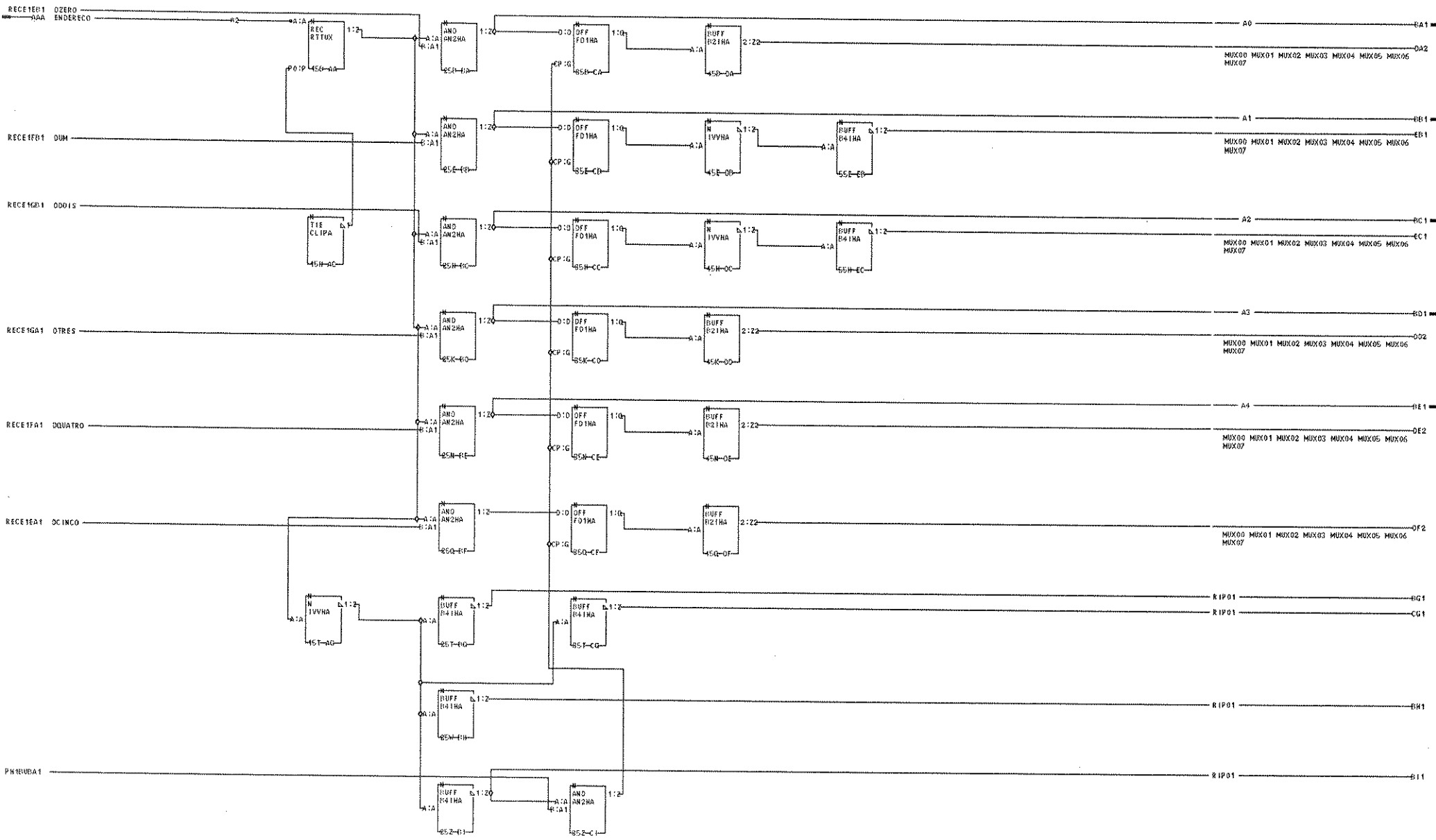
Observacao do conteudo do Registro p

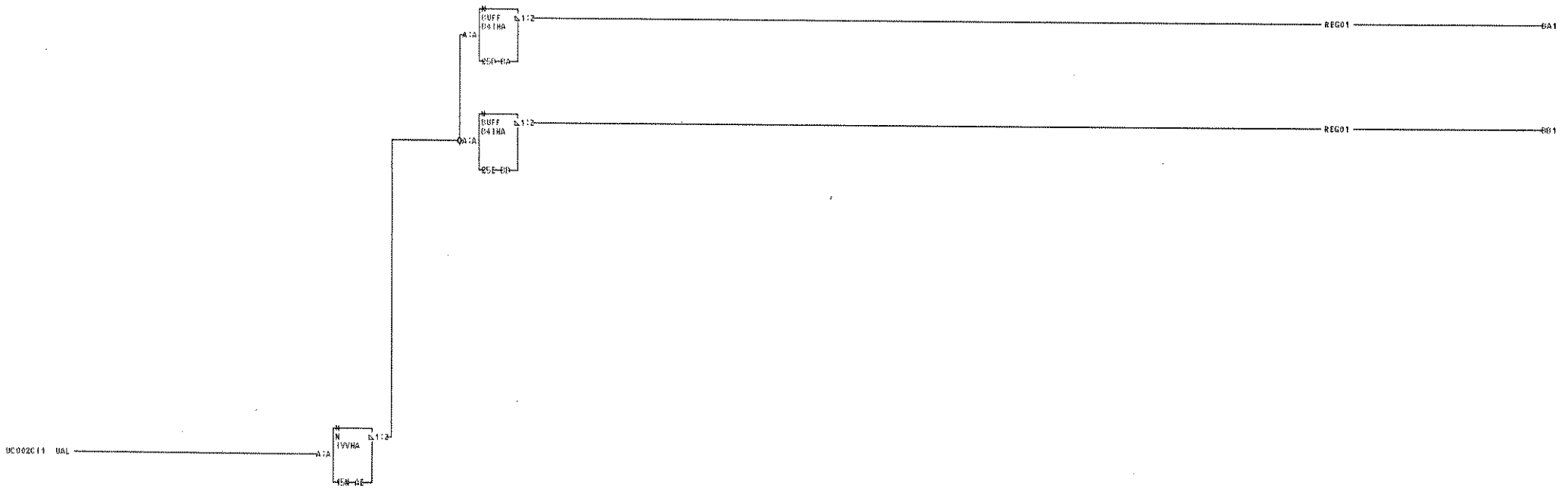
Para que um dado seja considerado endereco, sinal endereco deve ser ativado e carregar em D0-D5 o endereco do registro P desejado. E necessario - manter D0-D5 por pelo menos 100Ns apos sinal de controle seja desativado. Podemos observar a saida de P0.




0008 P(3)
0009 P(4)
0010 P(5)
0011 P(6)
0012 P(7)
0013 D(0)
0014 D(1)
0015 D(2)
0016 D(3)
0017 D(4)
0018 D(5)
0019 D(6)
0020 D(7)

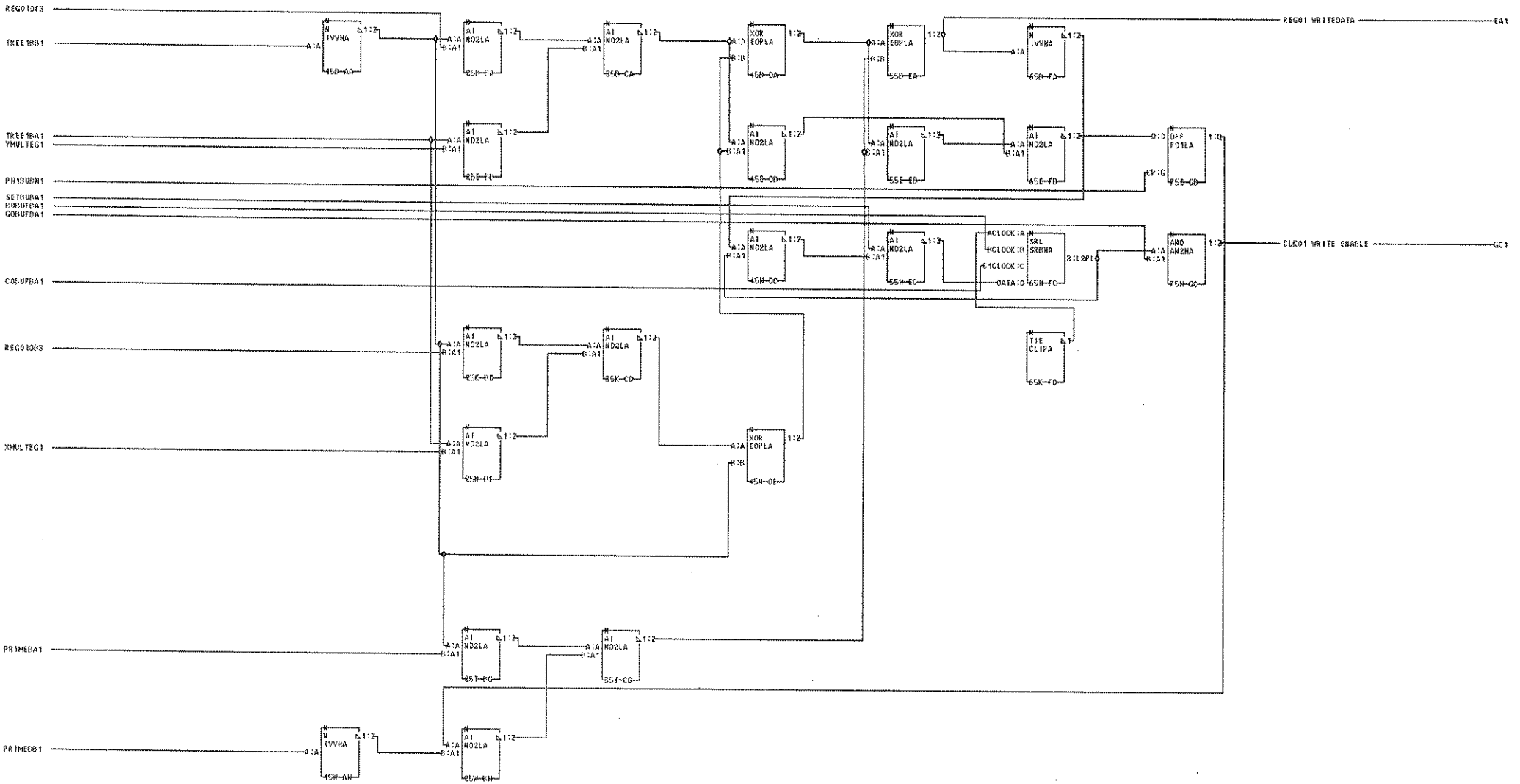

D0
D1
D2
D3
D4
D5
D6
D7
PH1
PH2
DADO
EDGE
RESET
ENDERECO
A30A30A
0:P0
01:P1
02:P2
03:P3
04:P4
05:P5
06:P6
07:P7
MUX01C01
MUX00C01
MUX02C01
MUX04C01
MUX06C01
MUX03C01
MUX05C01
MUX07C01


RECE1EB1 OZERO
AAA ENDERECO
REC RTTUX
AND AN2HA
DFF FD1HA
BUFF B21HA
A0
BA1
BA2
MUX00 MUX01 MUX02 MUX03 MUX04 MUX05 MUX06 MUX07
RECE1FB1 DUM
N IVVHA
BUFF B41HA
A1
BB1
BB2
RECE1GB1 ODOIS
TIE CLIPA
A2
BC1
BC2
RECE1GA1 OTRES
A3
BD1
BD2
RECE1FA1 DQUATRO
A4
BE1
BE2
RECE1EA1 QCINCO
BF1
BF2
RIP01
BG1
CG1
BH1
PNTRUBA1
BI1
CONNECTORS
AAA
0001/1A-A1A1A30A30AA01
ENTRADA DO ENDERECO XE
LOC=1A-A1A1A30A30AA01
USN 00001 PRI=020CT91 1342
AUC= SEC
PFORM= NEXTBLK ED
MACH=0BSL
CIO ZKOND JOB KONDORB
ADDRE 0001



BUFER PARA SINAL UAL	
LOC=1A-A1A1A30A30AA01	
USN 00001	PRI=020CT91 1342
AUC=	SEC
PFORM=	NEXTBLK 0C
MACHE=RSL	
CID ZKONG	JOB KONDOR8

A L U B U 0001

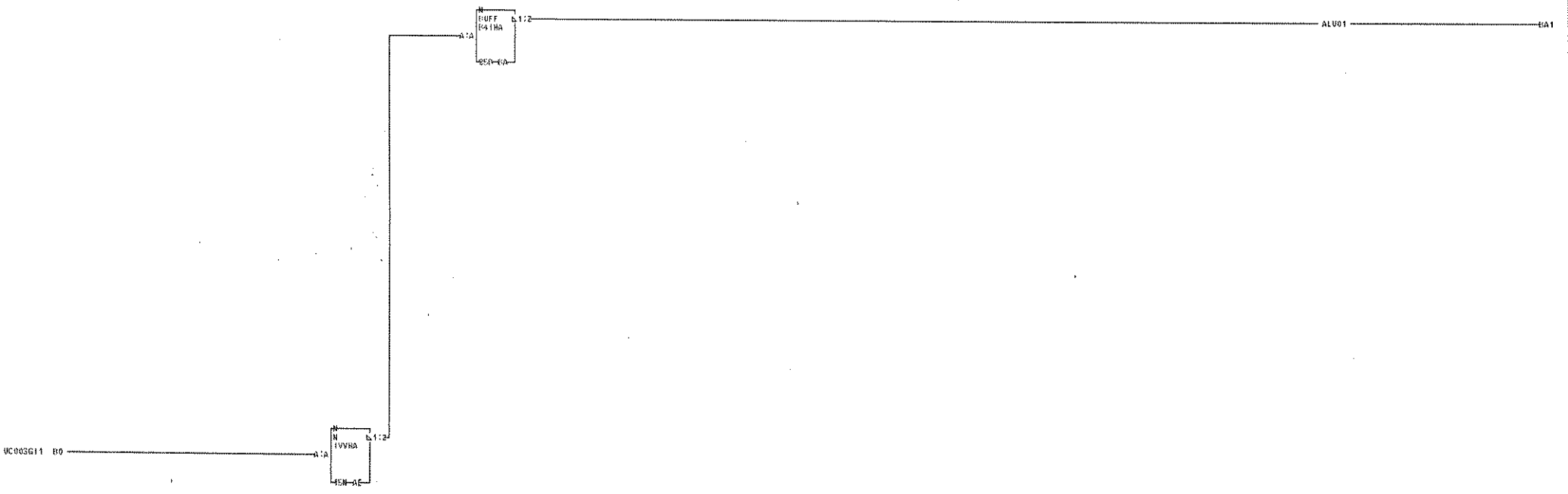


UAL0

LOC=1A-01A1A30A30AA01

USN 00001 PRI=02OCT91 1342

AUC= SEC
PFORM= NEXTBLK GD
MACH=IRSL
CID ZKOND JOB KONDORB

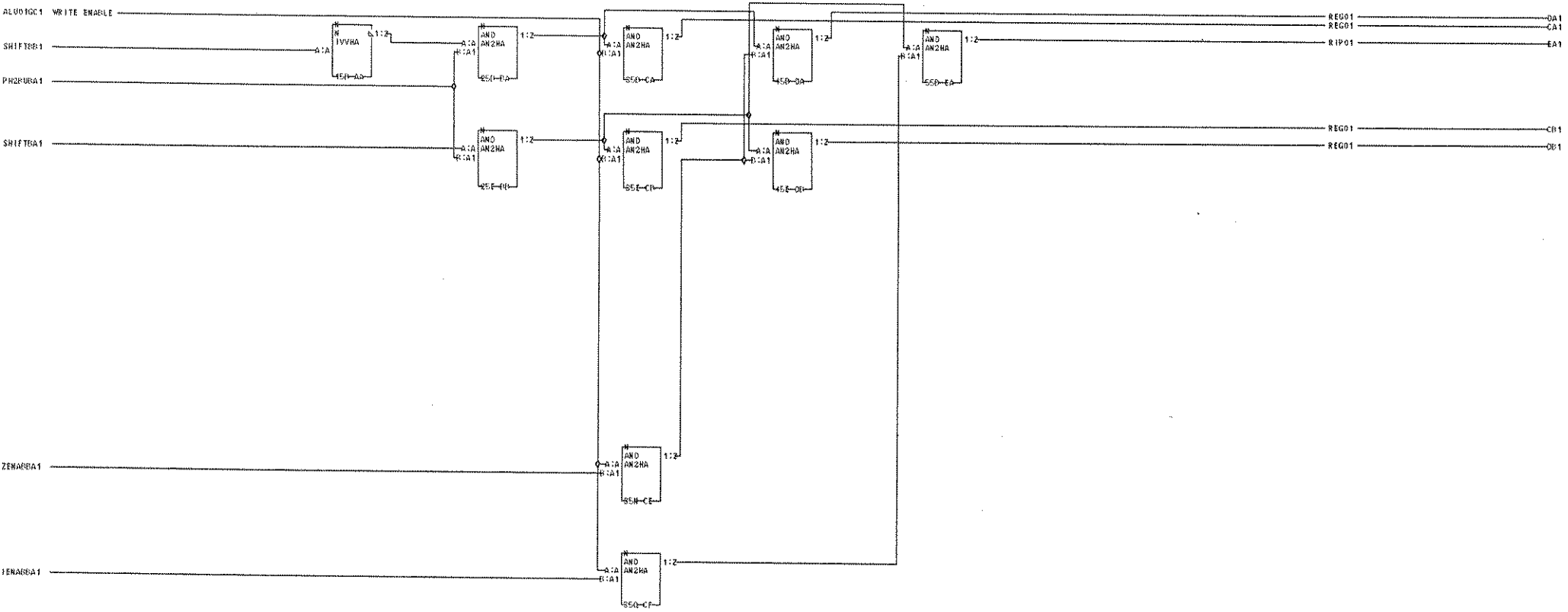


BUFFER PARA B0

LOC=1A-A1A1A30A30AA01

USN 00001 PRI=020CT91 1342

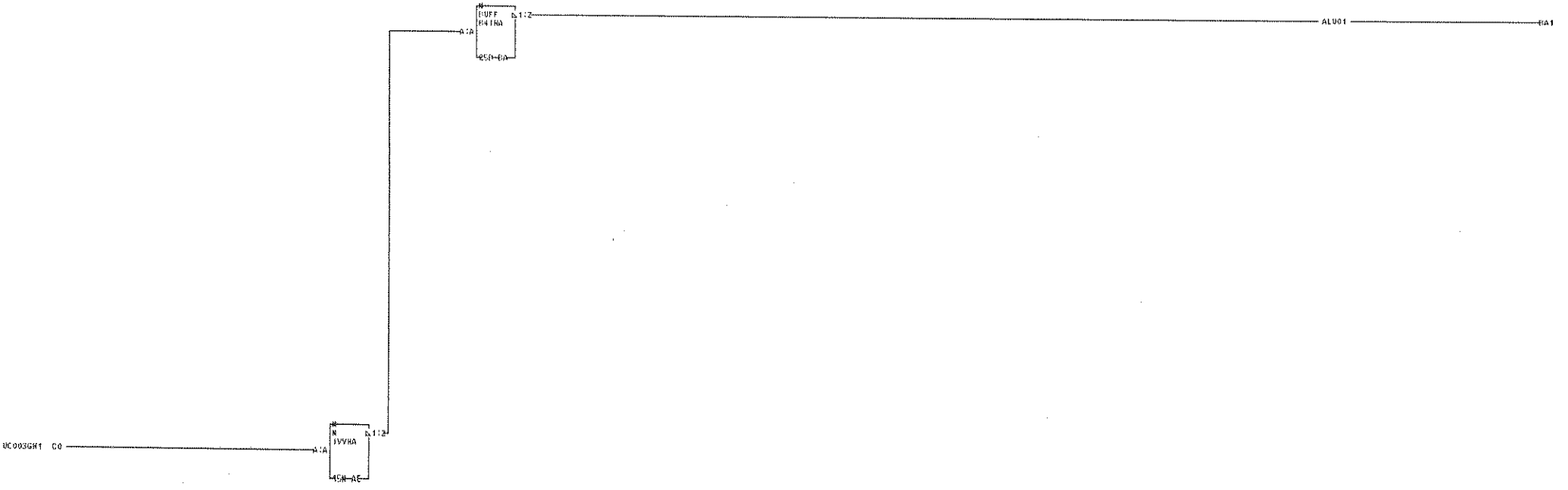
AUC= SEC
PFORM= NEXTBLK 88
MACH=SRSL
CIO ZKOND JOB KONDOBB



CLOCKS DE Z,F,I,PO	
LOC=1A-A1A1A30A30AA01	
DSN 00001	PRI=020CT91 1342
AUC=	SEC
PFORM=	NEXTCLK EB
MACH=ERSL	
CID ZKOND	JOB KONDORB

CLK01

0001

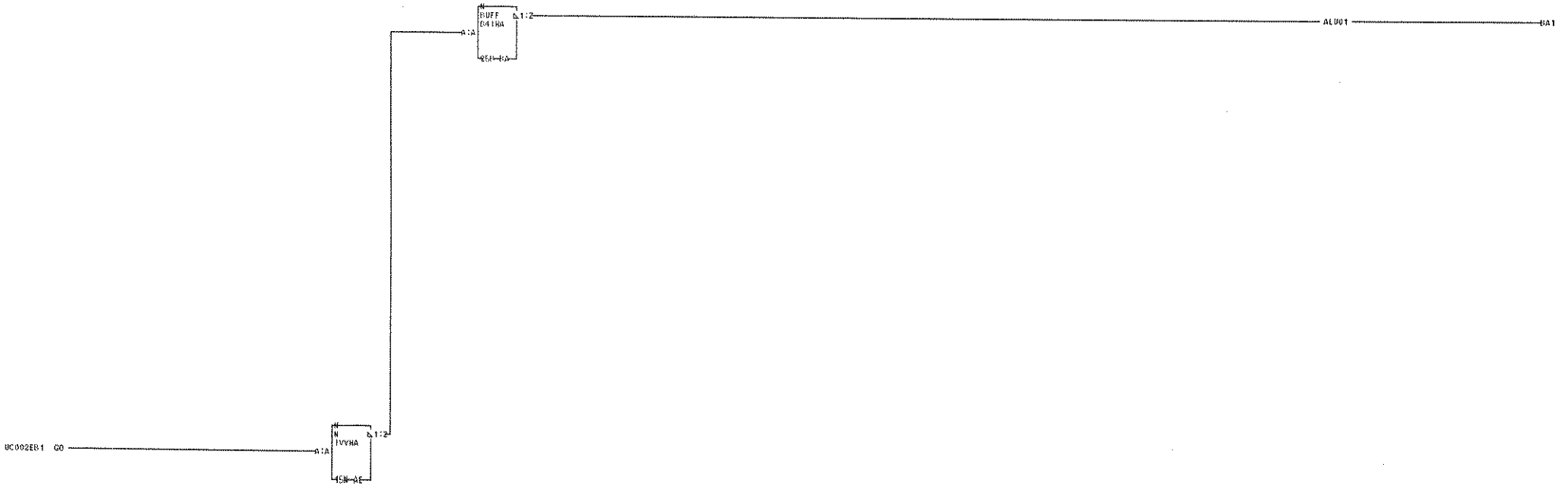


BUFFER PARA C0

LOC=1A-A1A1A30A30AA01

USN 00001 PRI=0200791 1342

AUC= SEC
PFORM= NEXTELK BB
MACH=BRSL
CID ZKOHD JOB KONOORB

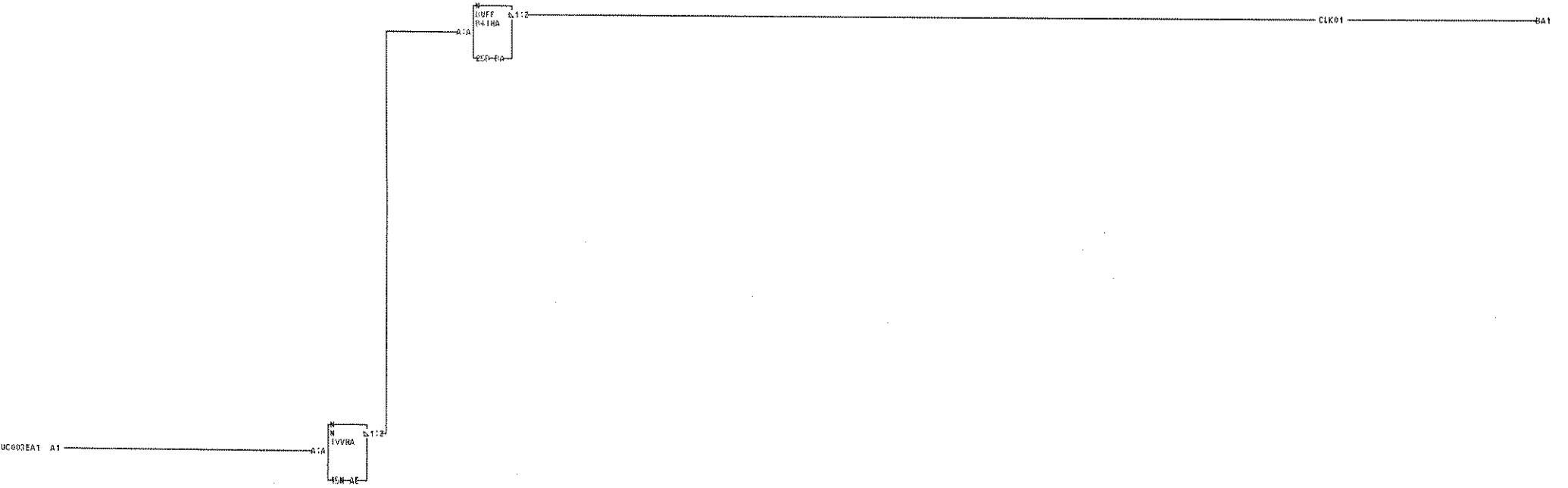


BUFFER PARA SINAL GO

LOC=1A-A1A1A30A30AA01

USN 00001 PRI=02OCT91 1342

AUC= SEC
PFORM= NEXTBLK BB
MACH=RISL
CID ZEONO JOB KONDORB



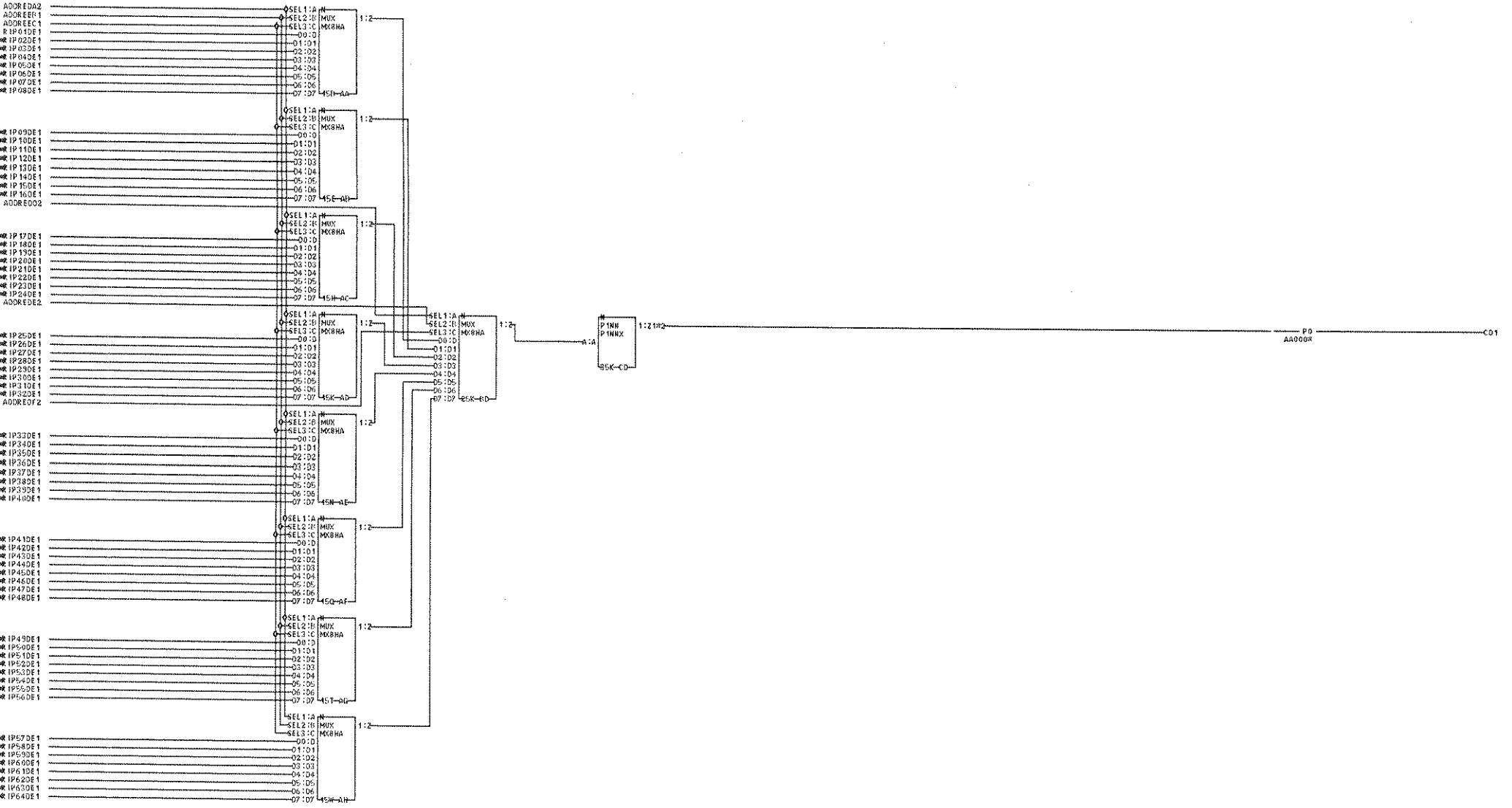
BUFFER DE HABILITA 1

LOC=1A-A1A1A30A30AA01

USN 00001 PRI=020CT91 1342

AUC=
PFORM=
MACHSERSL
CIO ZKOND

SEC
NEXTBLK BB
JOB KONDORB



CONNECTORS

CD1

0001/1A-A1A1A30A30AA01

MUX DO BIT ZERO DOS 64 REG P

LOC=1A-A1A1A30A30AA01

USN 00001 PRI=020CT91 1342

AUC= SEC

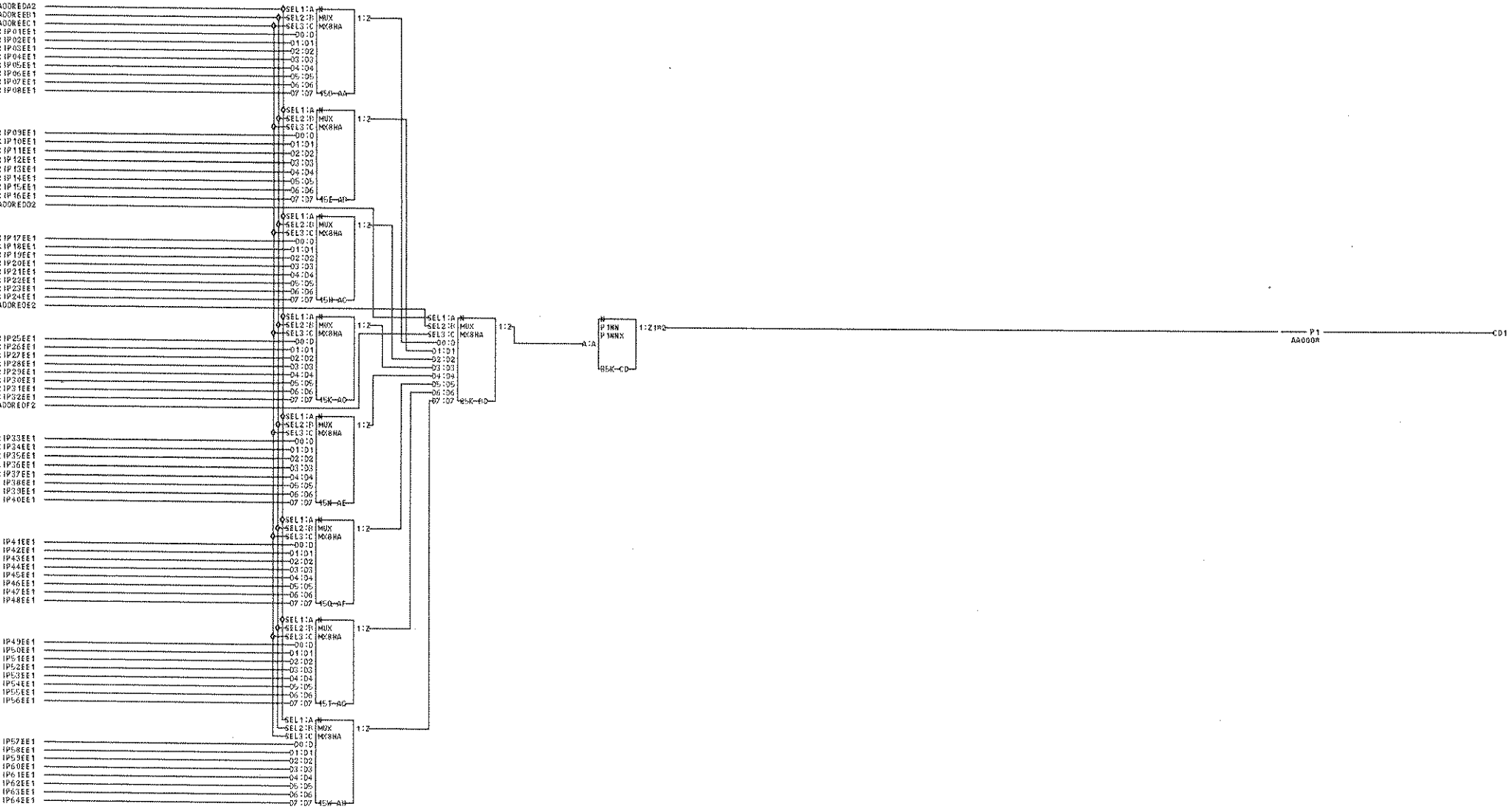
PFORMS NEXTBLK CE

MACHINRSL

CID ZKOND JOB KONDORB

MUX00 0001

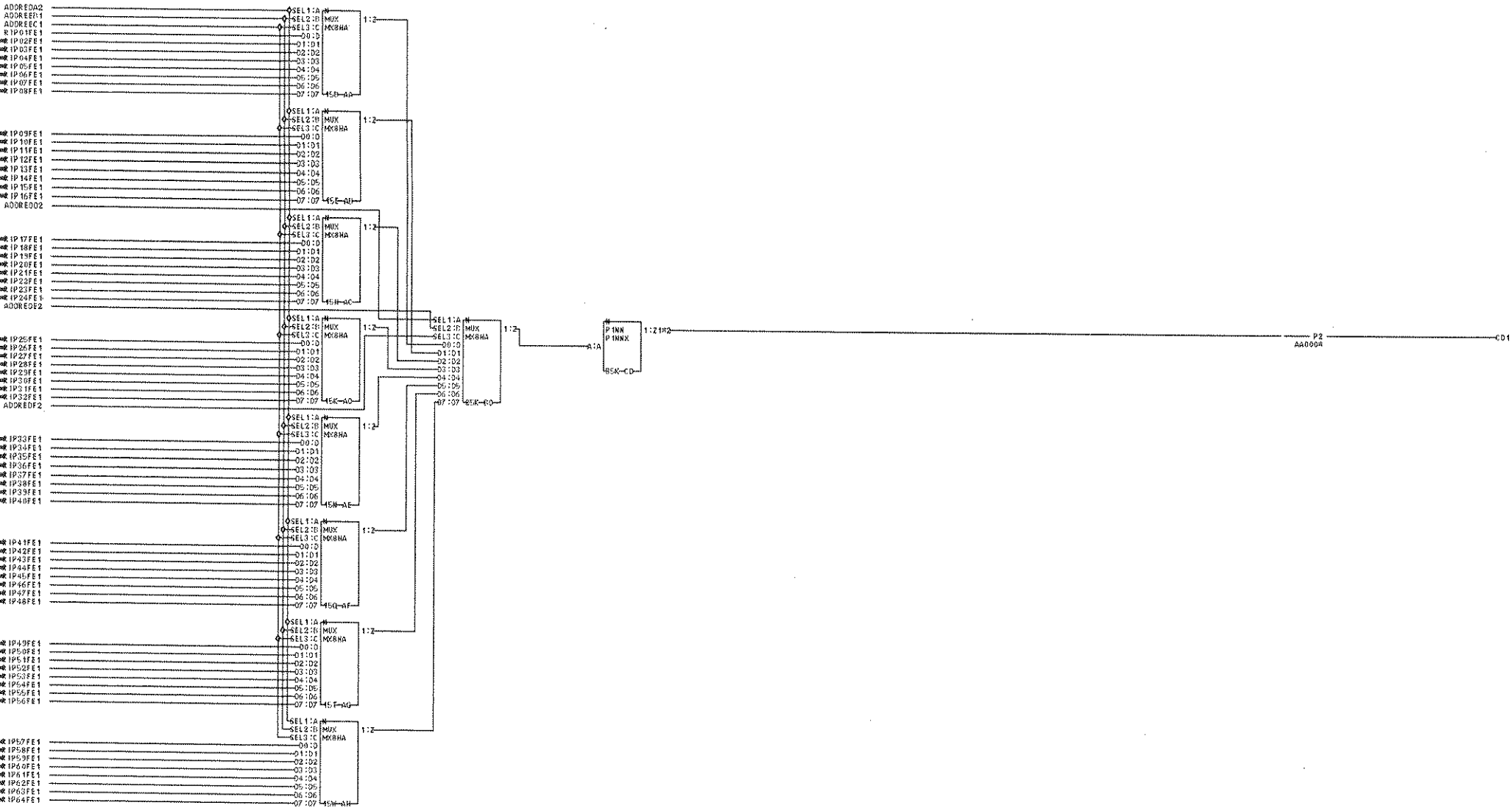
MUX00 0001



CONNECTORS
CD1
0001/1A-A1A1A30A30AA01

MUX 00 BIT UM DOS 64 REG P	
LOC=1A-A1A1A30A30AA01	
USN 00001	PRI=020CT91 1342
AUC=	SEC
PFORM=	NEXTBLK CE
MACHSERSL	
CID 2KOND	JOB KONDORB

MUX01
0001

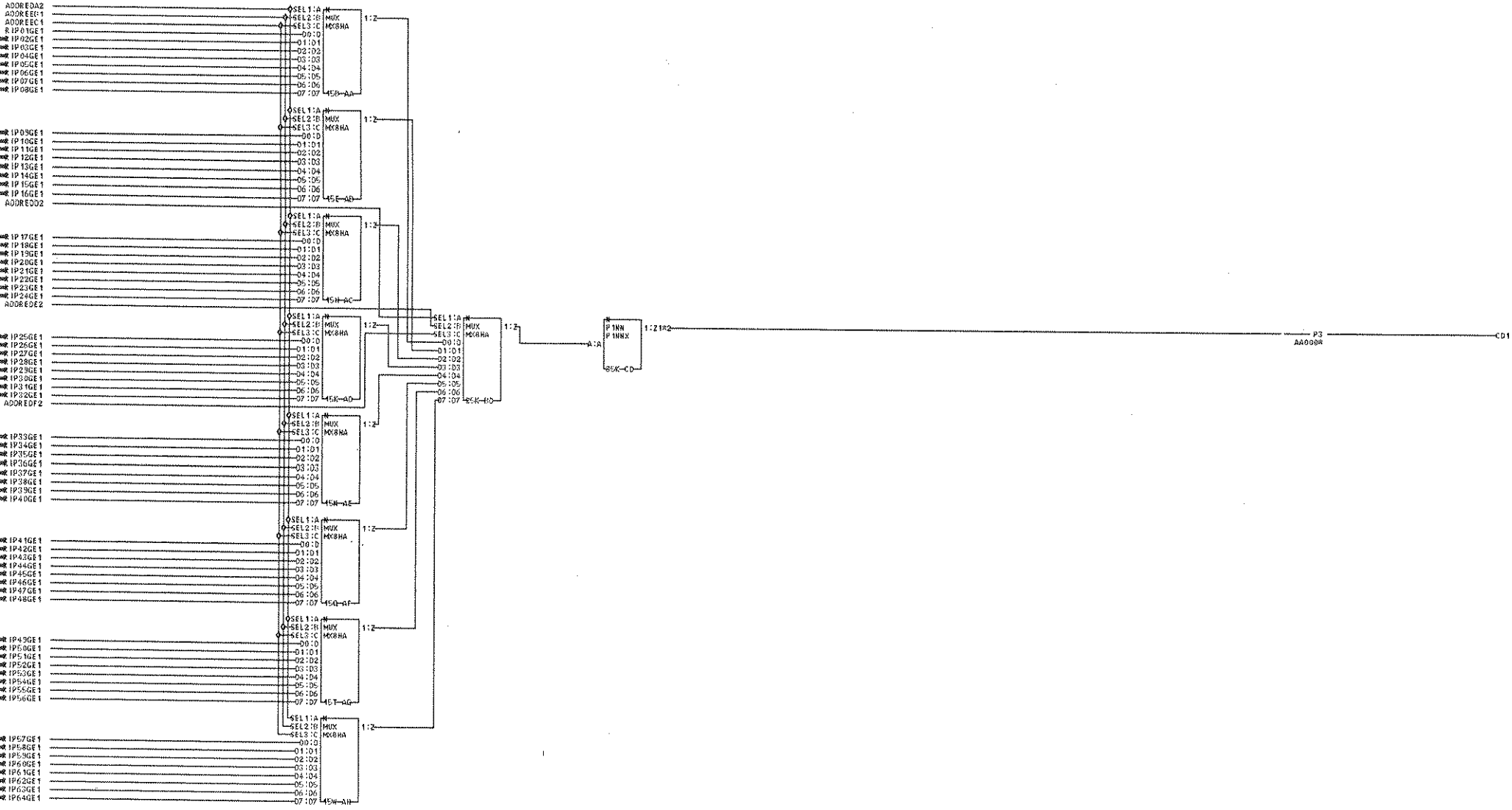


CONNECTORS
C01
0001/1A-A1A1A30A30AA01

MUX02 0001

MUX 00 BIT 0015 005 64 REG P	
LOC=1A-A1A1A30A30AA01	
USN 00001	PR1=020CT91 1342
AUC=	SEC
PFORM=	NEXTBLK CE
MACH=BRSL	
CID ZKOW3	JOB KONDORB

MUX02 0001



CONNECTORS

CD1

0001/1A-A1A1A30A30AA01

MUX03 0001

MUX DO BIT TRES DOS 64 REG P

LOC=1A-A1A1A30A30AA01

USN 00001 PRI=020CT91 1342

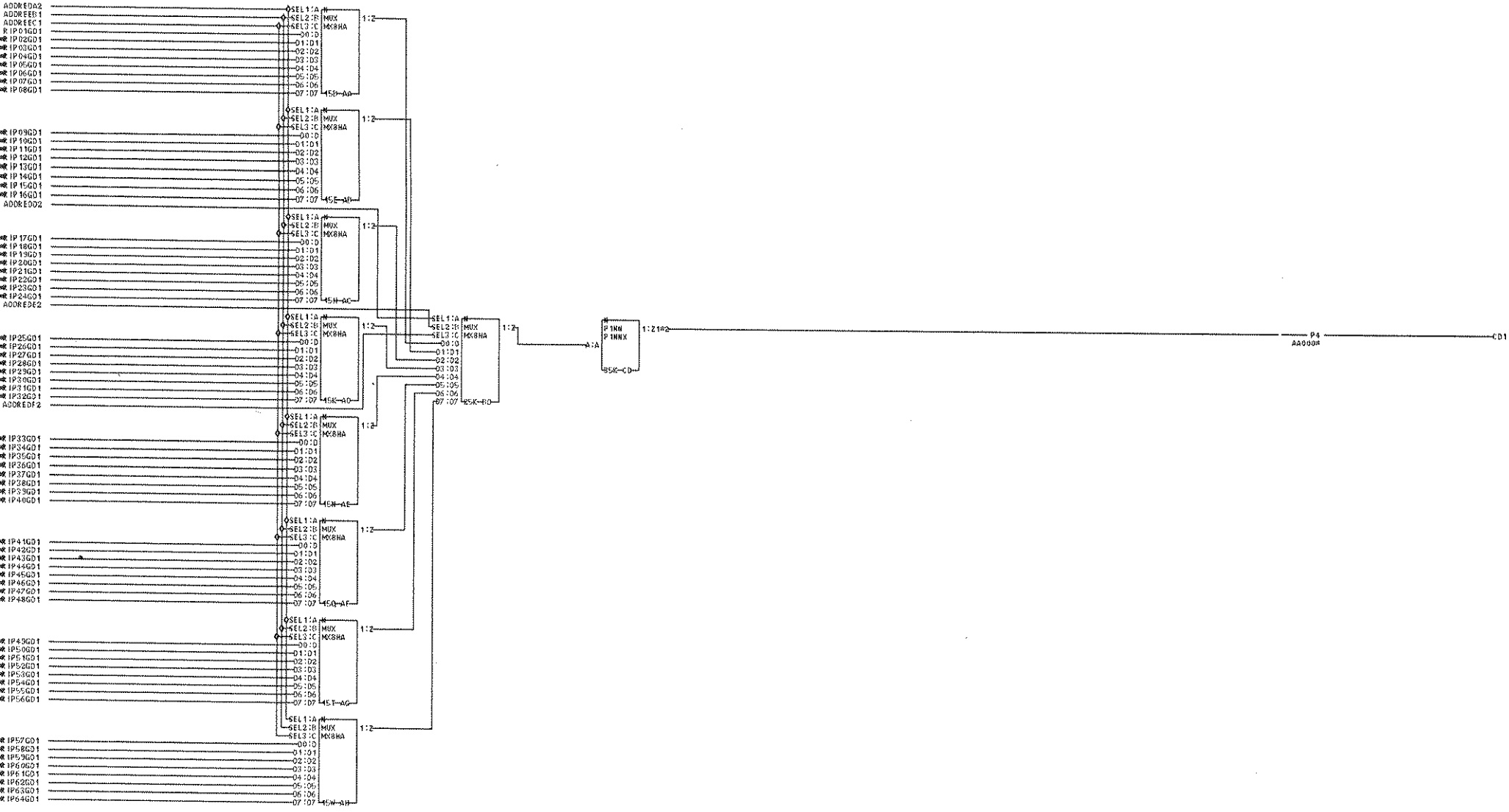
AUC= SEC

PFORM= NEXTBLK CE

MACH=BRSL

CID ZKONO JOB KONDOR8

MUX03 0001



CONNECTORS

C01

0001/1A-A1A1A30A30AA01

MUX 00 BIT QUATRO DOS 64 REG P

LOC=1A-A1A1A30A30AA01

USN 00001 PRI=020CT91 1342

AUC= SEC

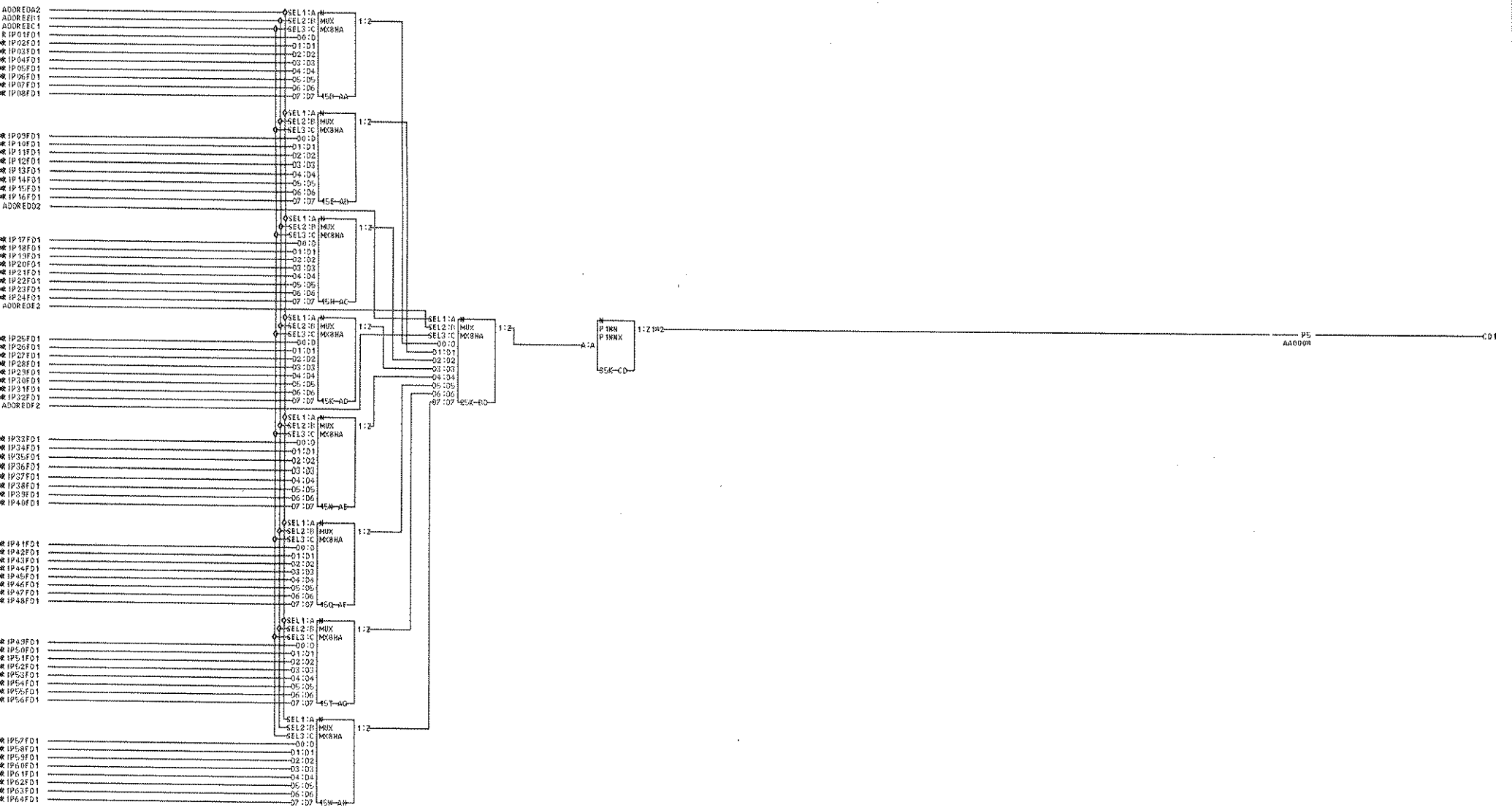
PFORM= NEXTBLK CE

MACH=DRSL

CIO ZKOND JOB KONDORB

MUX04

0001

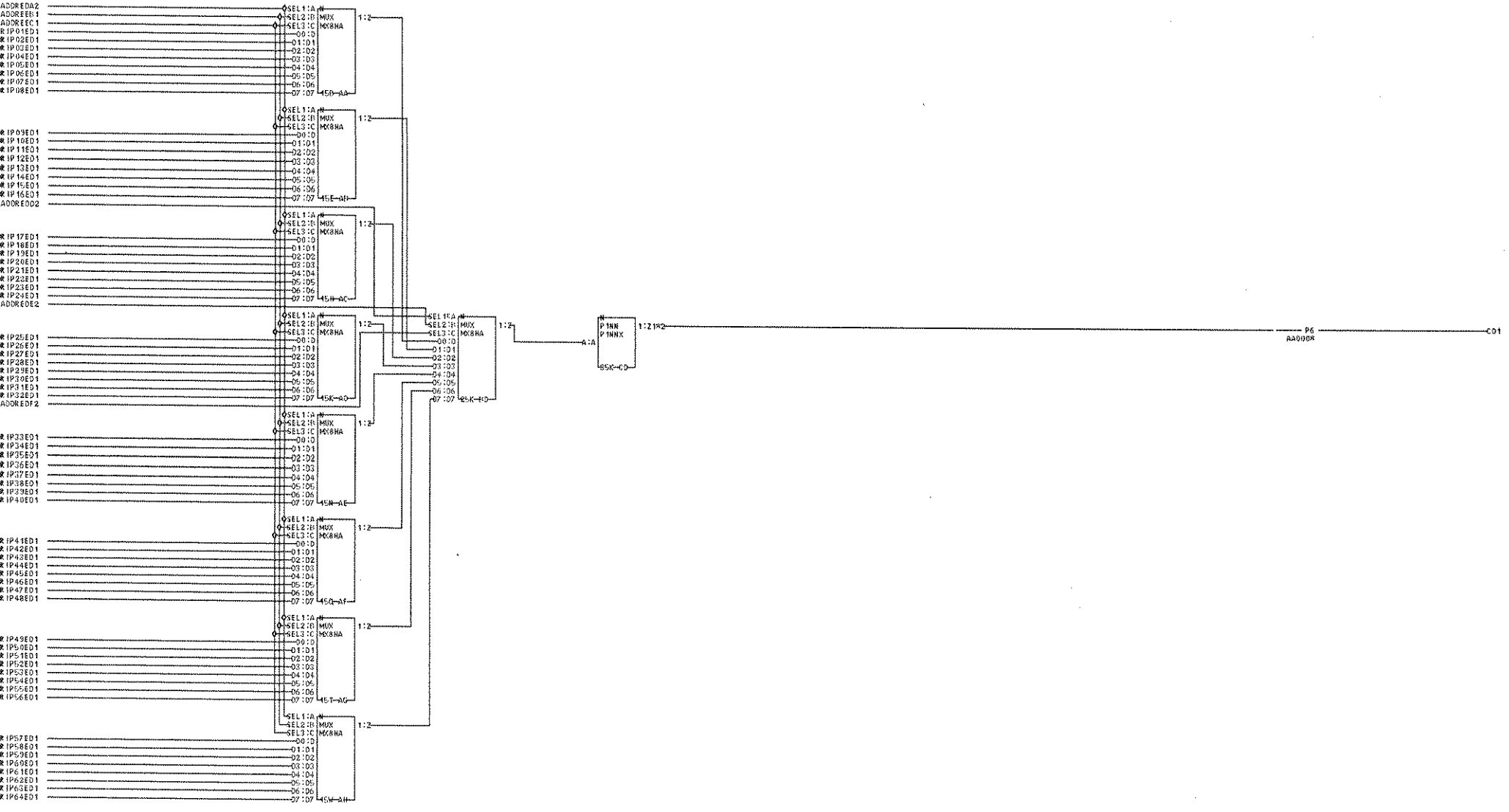


CONNECTORS

C01

0001/1A-A1A1A30A30AA01

MUX DO BIT CINCO DOS 64 REG P	
LOC=1A-A1A1A30A30AA01	
USN 00001	PR1=020CT91 1342
AUC=	SEC
PFORM=	NEXTBLK CE
MACH=BRSL	
CID 2KOND	JOB KONDOR0



CONNECTORS
CO1
0001/1A-A1A1A30A30AA01

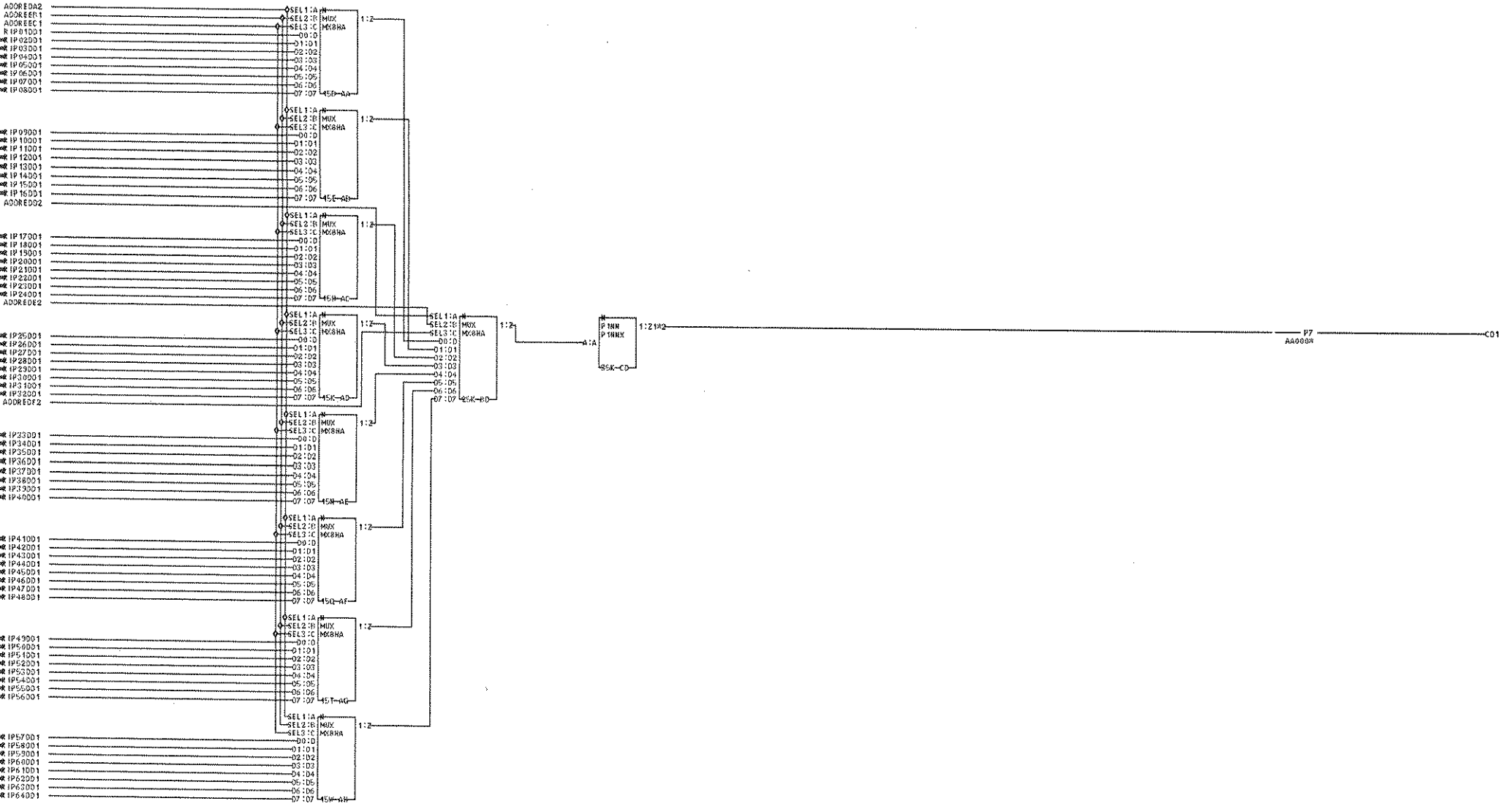
MUX 00 BIT SEIS 005 64 REG P

LOC=1A-A1A1A30A30AA01

USN 00001 PRI=02OCT91 1342

AUC= SEC
PFORM= NEXTBLK CE
MACH=PRSL
CID ZKOND JOB KONDORB

0001



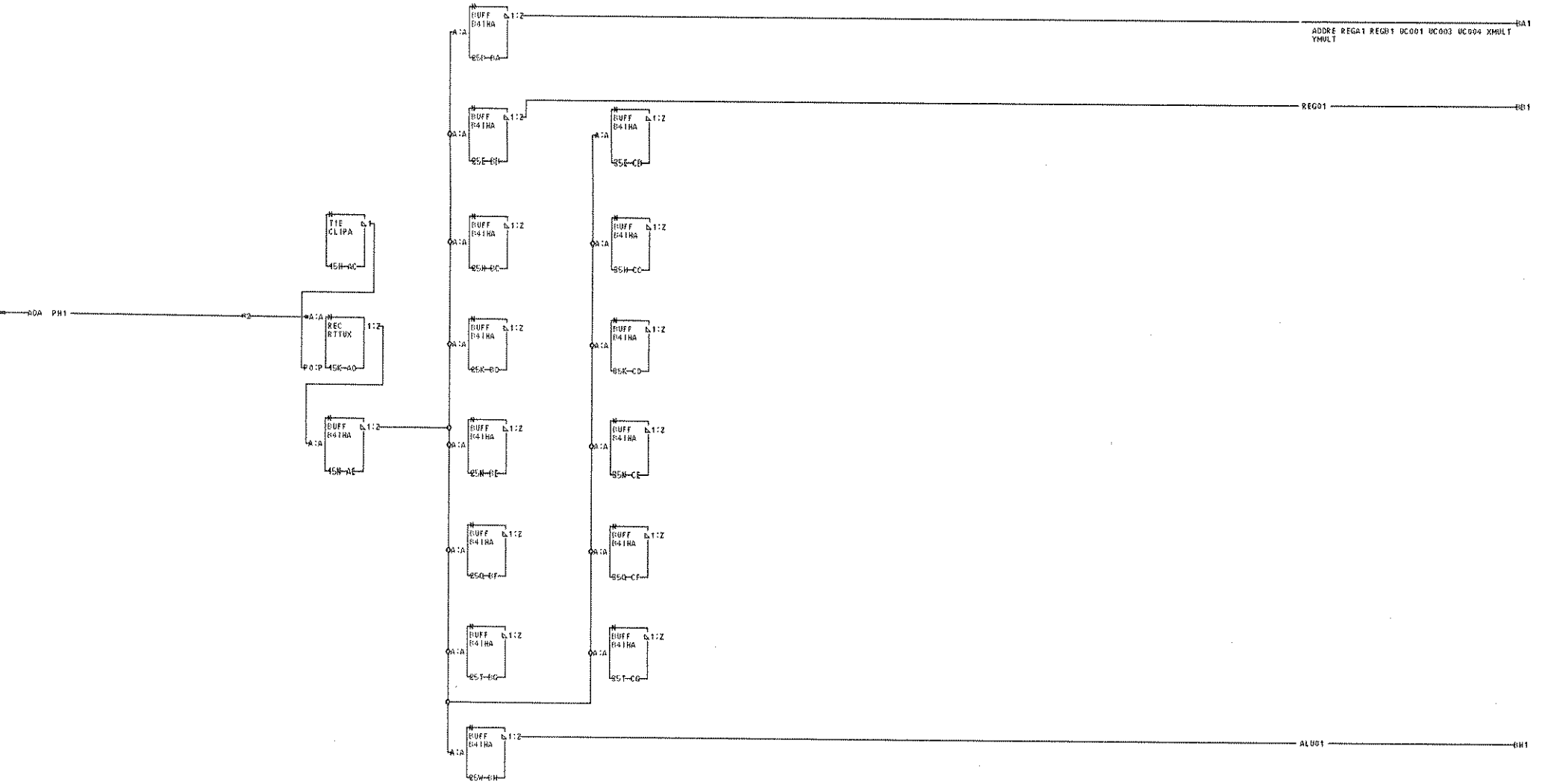
CONNECTORS

C01

0001/1A-A1A1A30A30AA01

MUX DO BIT SETE DOS 64 REG P	
LOC=1A-A1A1A30A30AA01	
USN 00001	PRT=02OCT91 1342
AUC=	SEC
PFORM=	NEXTCLK CE
MACH=BRSL	
CIO ZKOND	JOB KONDORB

MUX07 0001



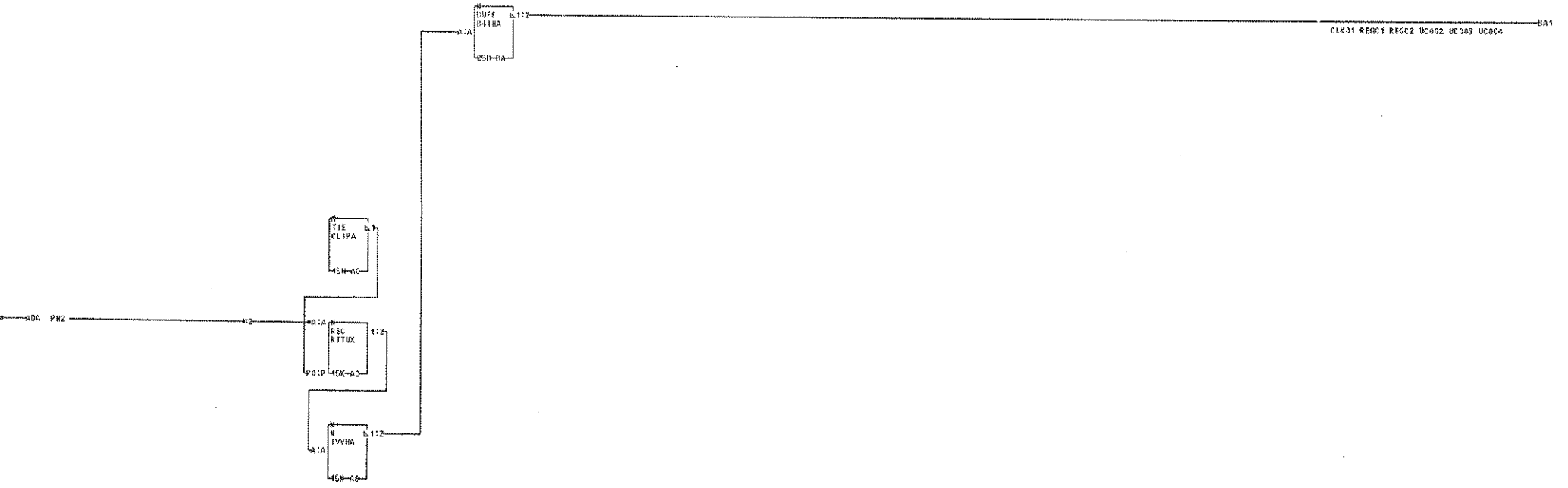
CONNECTORS
ADA
0001/1A-A1A1A30A30AA01

BUFER PARA CLOCK PH1

LOC=1A-A1A1A30A30AA01

USN 00001 PRT=02OCT91 1342

AUC= SEC
PFORM= NEXTBLK CH
MACH=BRSL
CFD 2KONO JOB KONDORB



CONNECTORS
ADA
0001/1A-A1A1A30A30AA01

BUFFER PARA CLOCK PH2

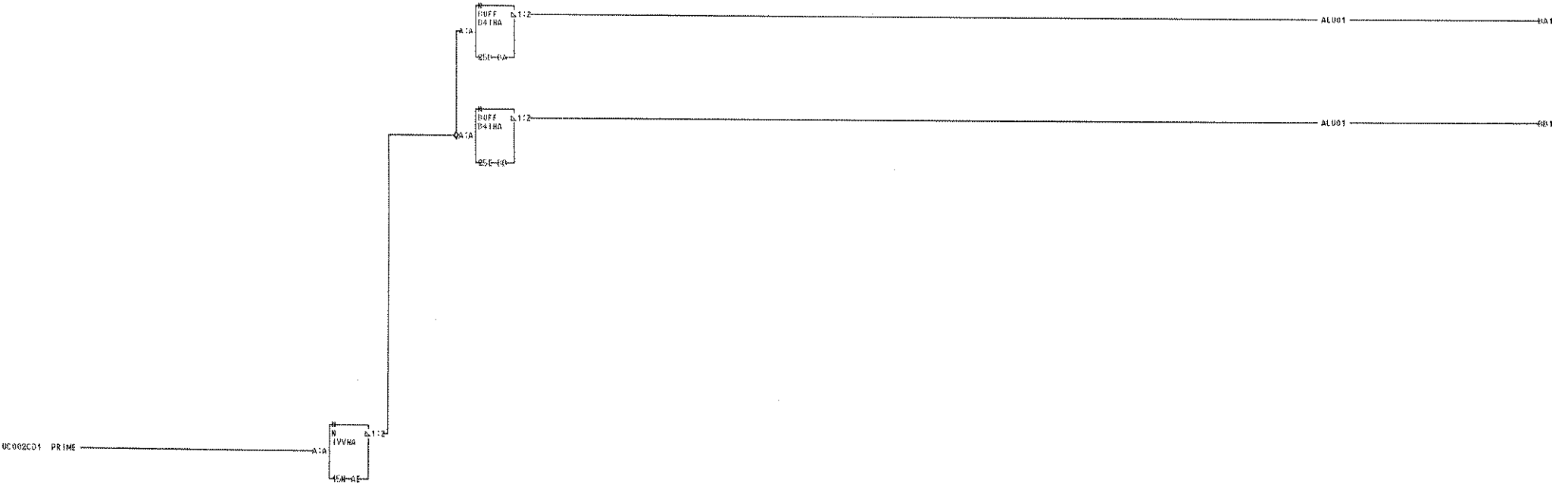
LOC=1A-A1A1A30A30AA01

USN 00001 PRI=02OCT91 1342

AUC= SEC
PFORM= NEXTBLK BB
MACH=IRSL
CID ZKOND JOB KONDORB

PH2 U

0001



BUFFER PARA SINAL PRIME

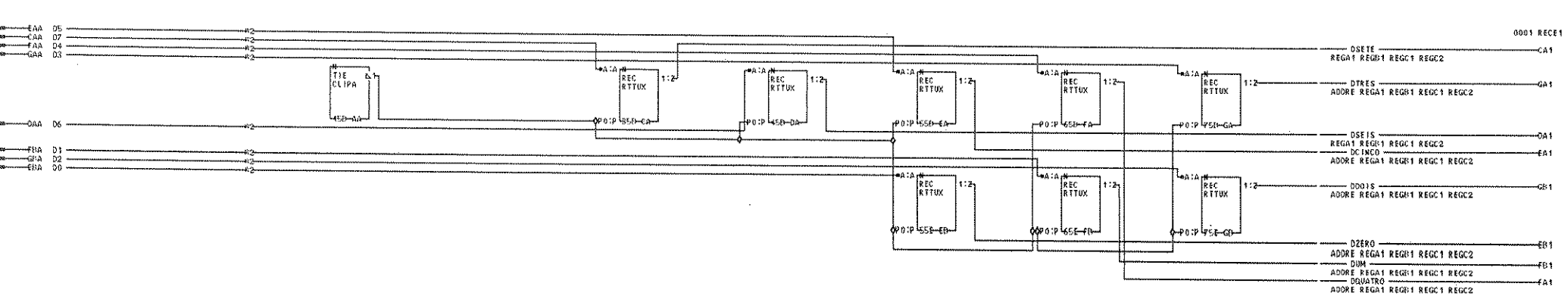
LOC=1A-A1A1A30A30AA01

USN 00001 PRI=020C791 1342

AUC=
PFORM=
MACH=ERSL
CIO ZKOND

SEC
NEXTBLK BC

JOB KONDORS



CONNECTORS
CAA
0001/1A-A1A1A30A30AA01
OAA
0002/1A-A1A1A30A30AA01
EAA
0003/1A-A1A1A30A30AA01
EPA
0004/1A-A1A1A30A30AA01
FAA
0005/1A-A1A1A30A30AA01
FPA
0006/1A-A1A1A30A30AA01
GAA
0007/1A-A1A1A30A30AA01

CONNECTORS
GBA
0008/1A-A1A1A30A30AA01

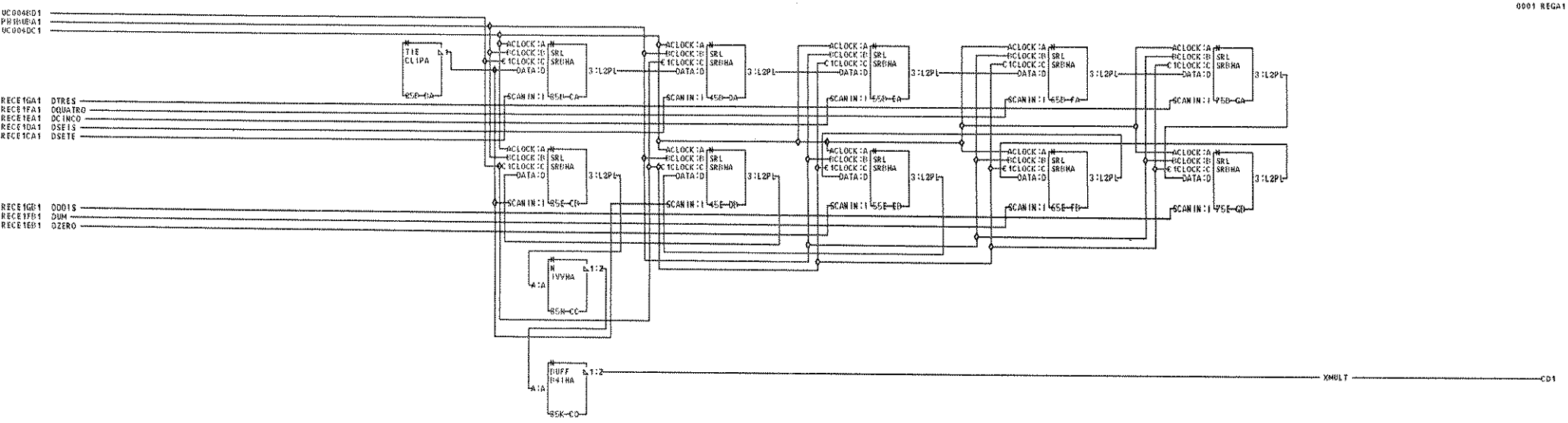
RECEPTORES DE ENTRADA

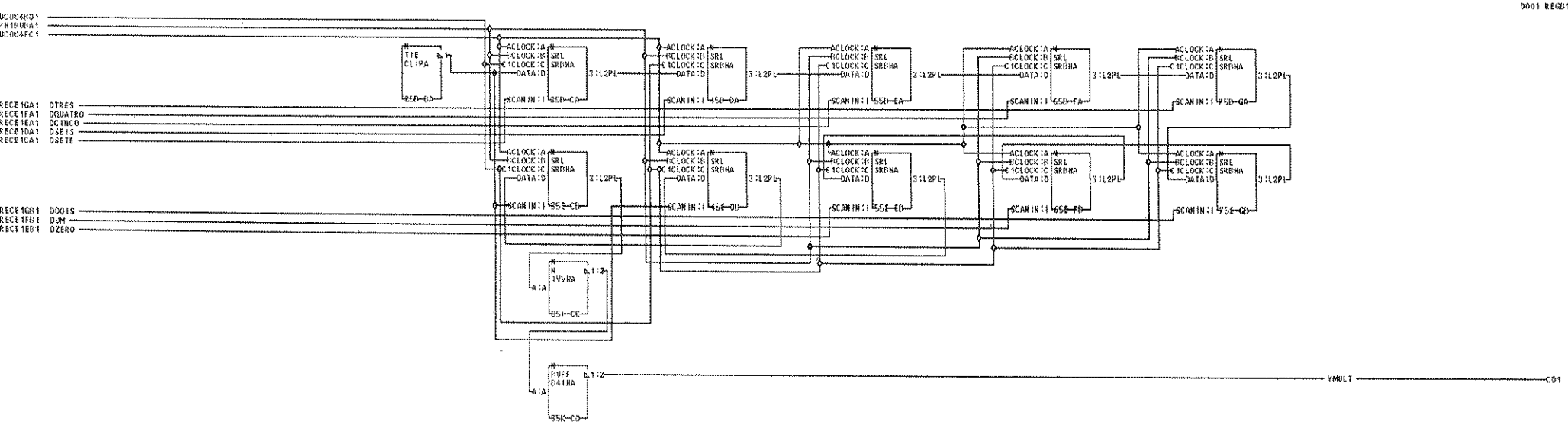
LOC=1A-A1A1A30A30AA01

USN 00001 PRI=020C791 1342

AUC= SEC
PFORM= NEXTBLK GC
MACH=PRSL
CID ZKOND JOB KONDORB

0001


UC004RD1
PRSHUEA1
UC004DC1
RECE1GA1 DTRES
RECE1FA1 DQUATRO
RECE1EA1 DCINCO
RECE1DA1 DSEIS
RECE1CA1 DSETE
RECE1GB1 DDOIS
RECE1FB1 DUM
RECE1EB1 DZERO
TIE CL1PA
ACLOCK:A
BCLOCK:B
C1CLOCK:C
DATA:D
SRL SRBHA
3:L2PL
SCANIN:I
65B-GA
65B-FA
65B-EA
65B-DA
65E-GB
65E-FB
65E-EB
65E-DB
65E-CB
65H-CA
65H-CC
65K-CD
N IVVBA
BUFF B41HA
A:A
B:Z
XMULT
CD1



REGISTRO DE ARMAZENAMENTO B

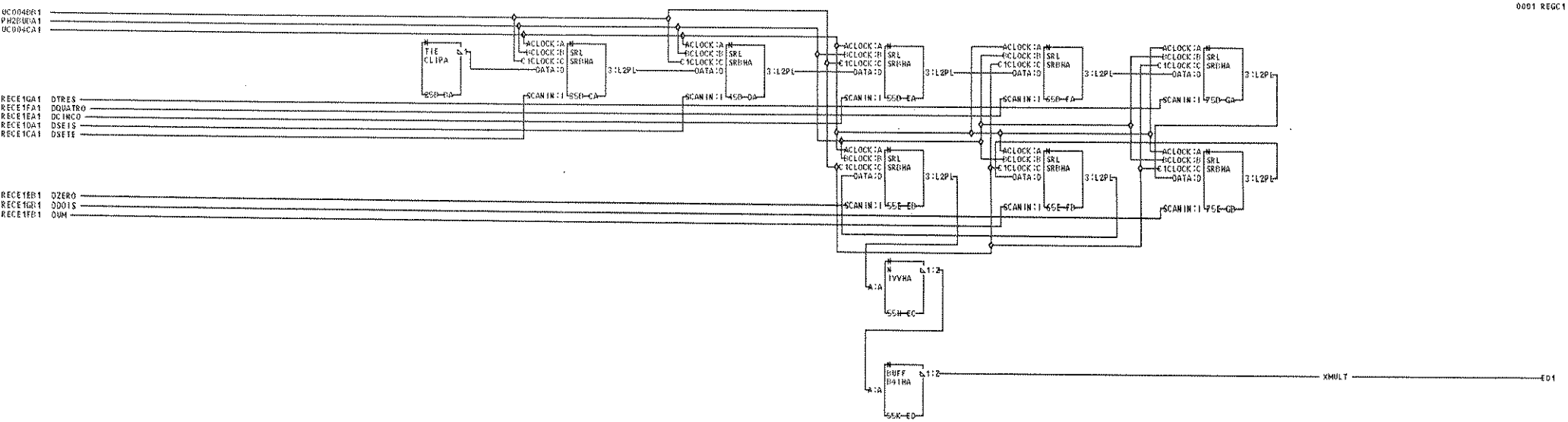
LOC=1A-A1A1A30A30AA01

USN 00001 PR1=02OCT91 1342

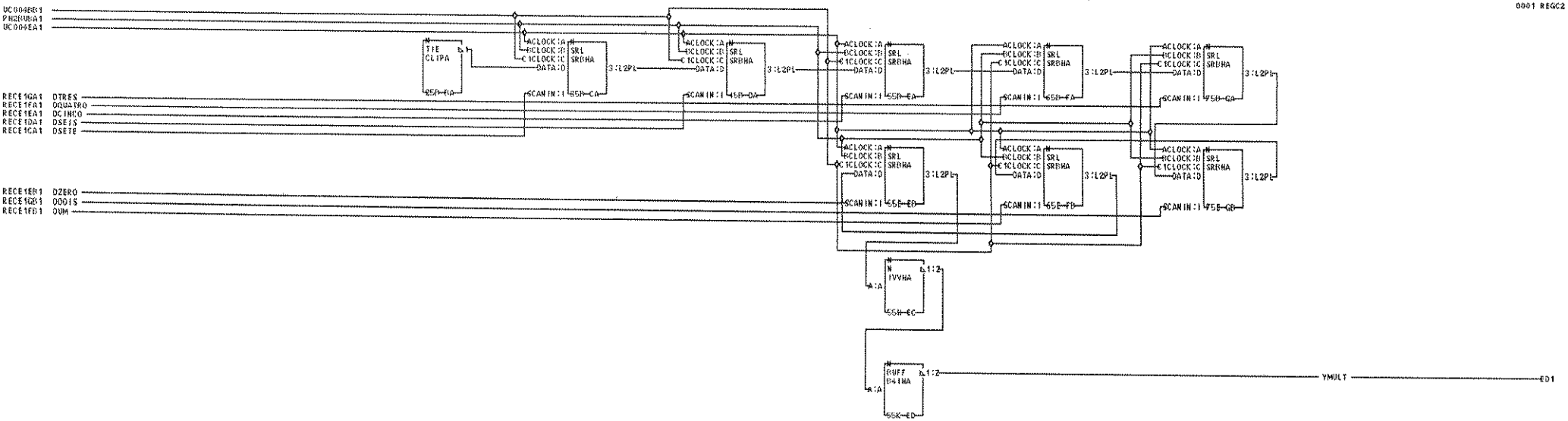
AUC= SEC
PFORM= NEXTBLK GC
MACH=RRSL
EC10 2KOND JOB KONDORB

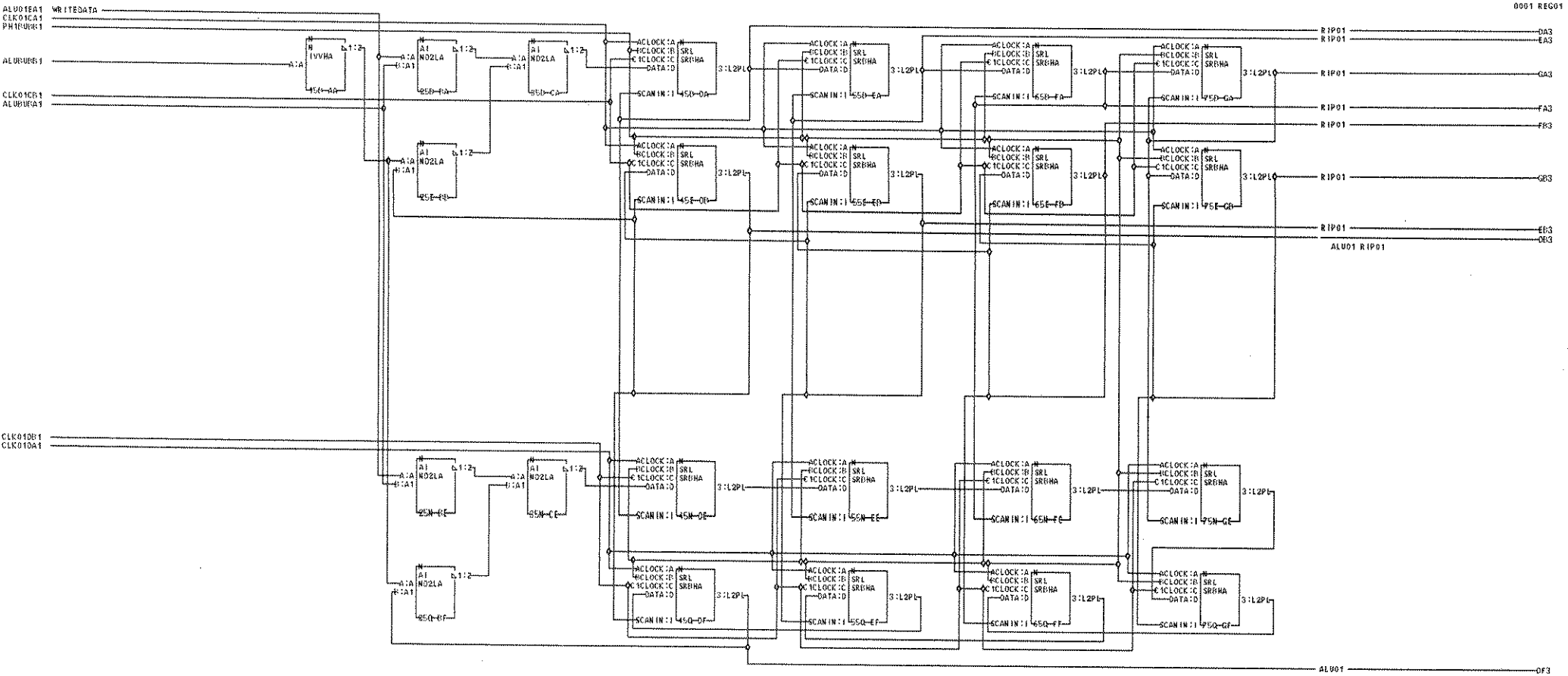
REG01

0001



REGISTRO DE ARMAZENAMENTO C1 1	
LOC=1A-A1A1A30A30AA01	
USN 00001	PRI=020CT91 1342
AUC=	SEC
PFORM=	NEXTBLK GC
MACHDRSL	
CIO ZKOND	JOB KOMOORB


UC004ER1
PRGNUMA1
UC004EA1
RECE1GA1 DTRES
RECE1FA1 DQUATRO
RECE1EA1 DCINCO
RECE1DA1 DSEIS
RECE1CA1 DSETE
RECE1ER1 DZERO
RECE1GB1 DOOIS
RECE1FB1 DUM
TIE CLIPA
SRL SRBHA
3:L2PL
ACLOCK:A
BCLOCK:B
CCLOCK:C
DATA:D
SCANIN:I
N FVVHA
BUFF D4IHA
YMULT
ED1



REGISTRADORES Z E F0

LOC=1A-A1A1A30A30AA01

USN 00001 PR1=02OCT91 1342

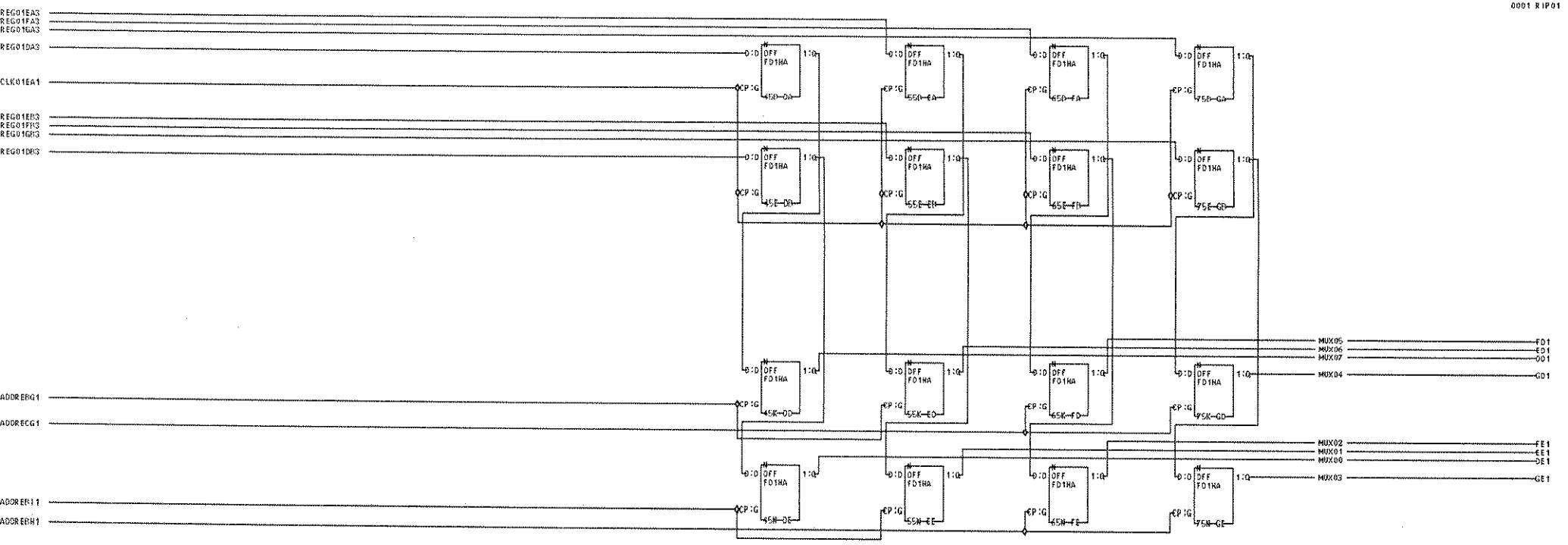
AUC= SEC

PFORM= NEXTBLK GG

MACN=DRSL

CIO ZKOND0 JOB KONDOR8

0001



REGISTRADORES P E IO

LOC=1A-A1A1A30A30AA01

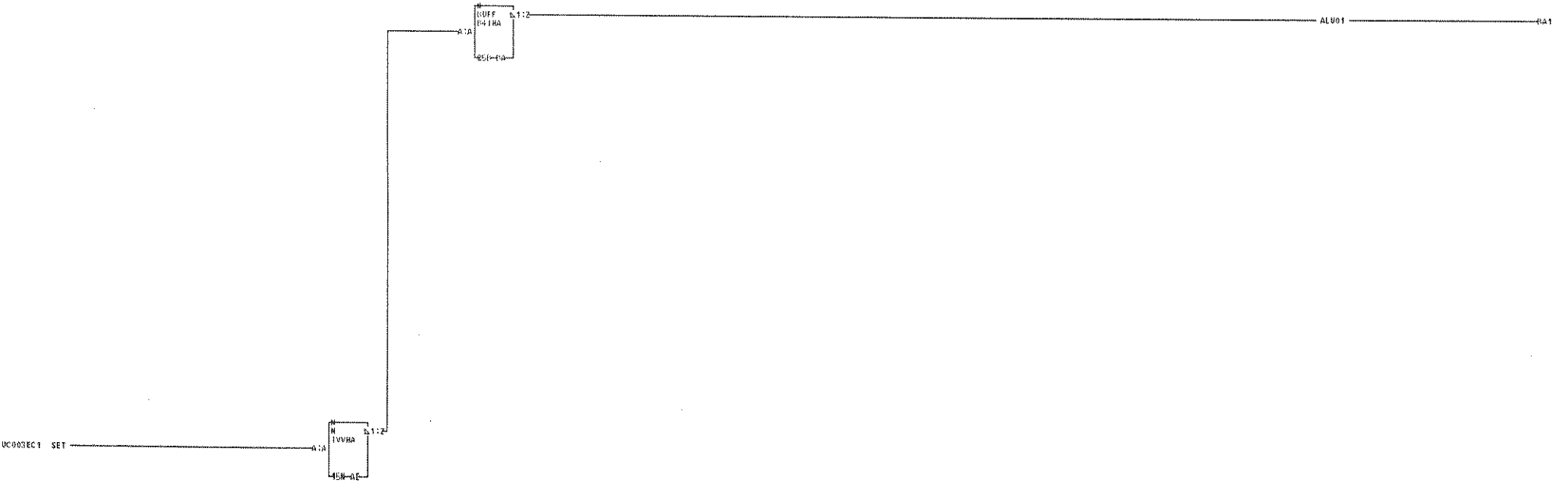
USN 00001 PR1=020C791 1342

AUC= SEC

PFORM= NEXTBLK GF

MACH=PRSL

CIO ZKOND JOB KONDORB



SETSU
001

BUFFER PARA SINAL SET

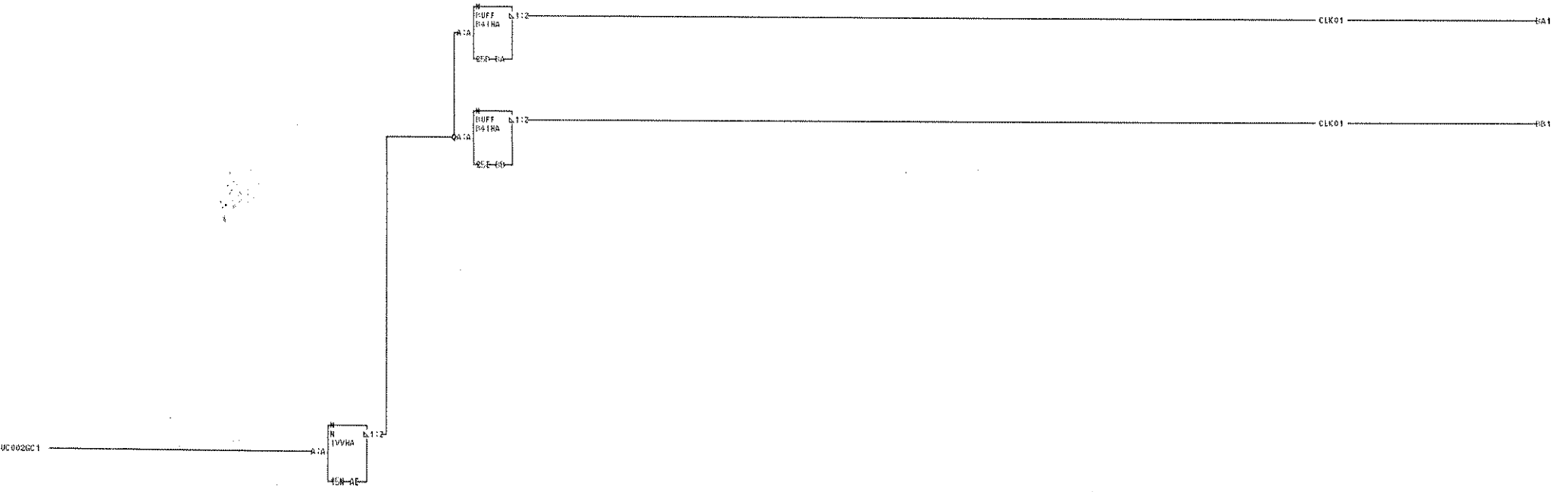
LOC=1A-A1A1A30A30AA01

USN 00001 PRI=020CT91 1342

AUC=
PFORMS
MACHINRSL
CIO ZKONO

SEC
NEXTBLK BB
JOB KONDORB

SETSU
0001



BUFFER PARA SINAL SHIFT

LOC=1A-A1A1A30A30AA01

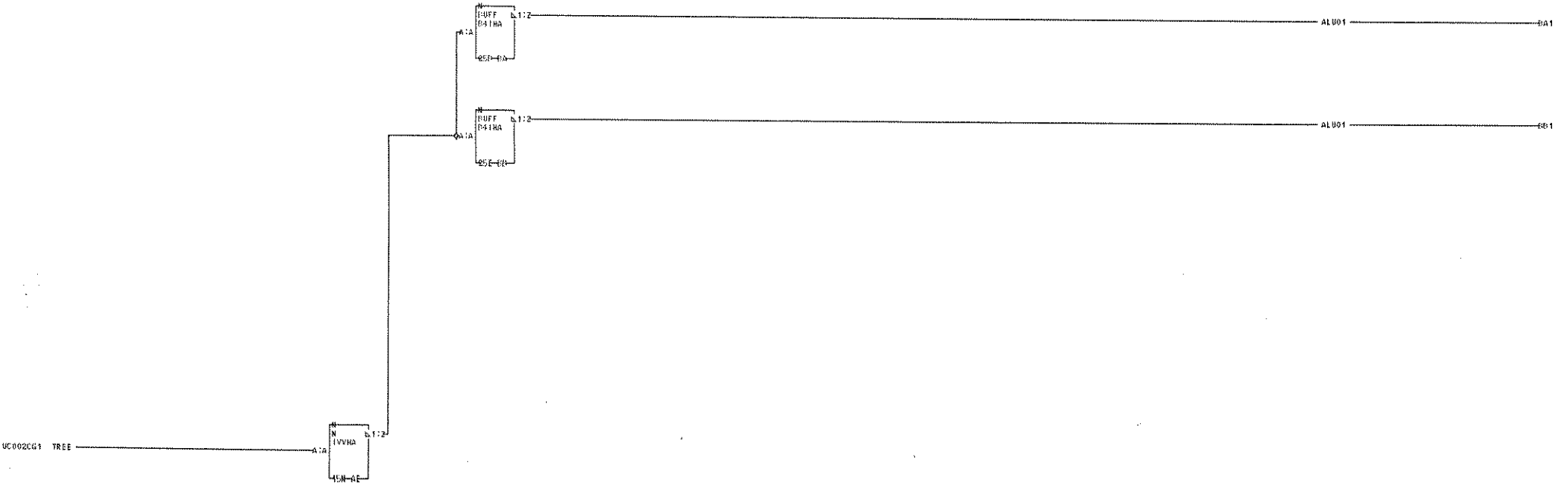
USN 00001 PRI=020CT91 1342

AUC=
PFORM=
MACHINE(RSL
CIO ZKONO

SEC
NEXTBLK GC

JOB KONDORB

0001

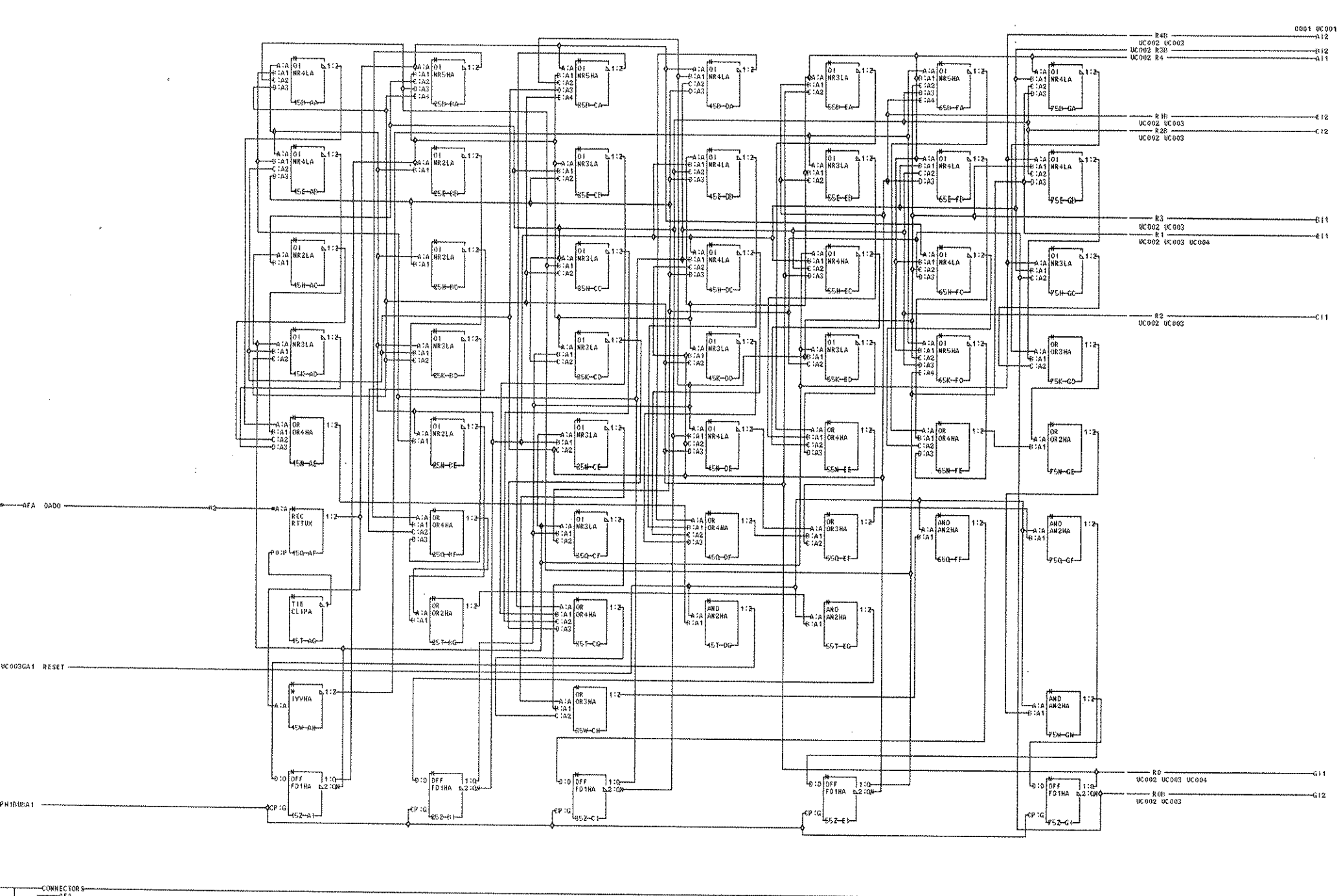


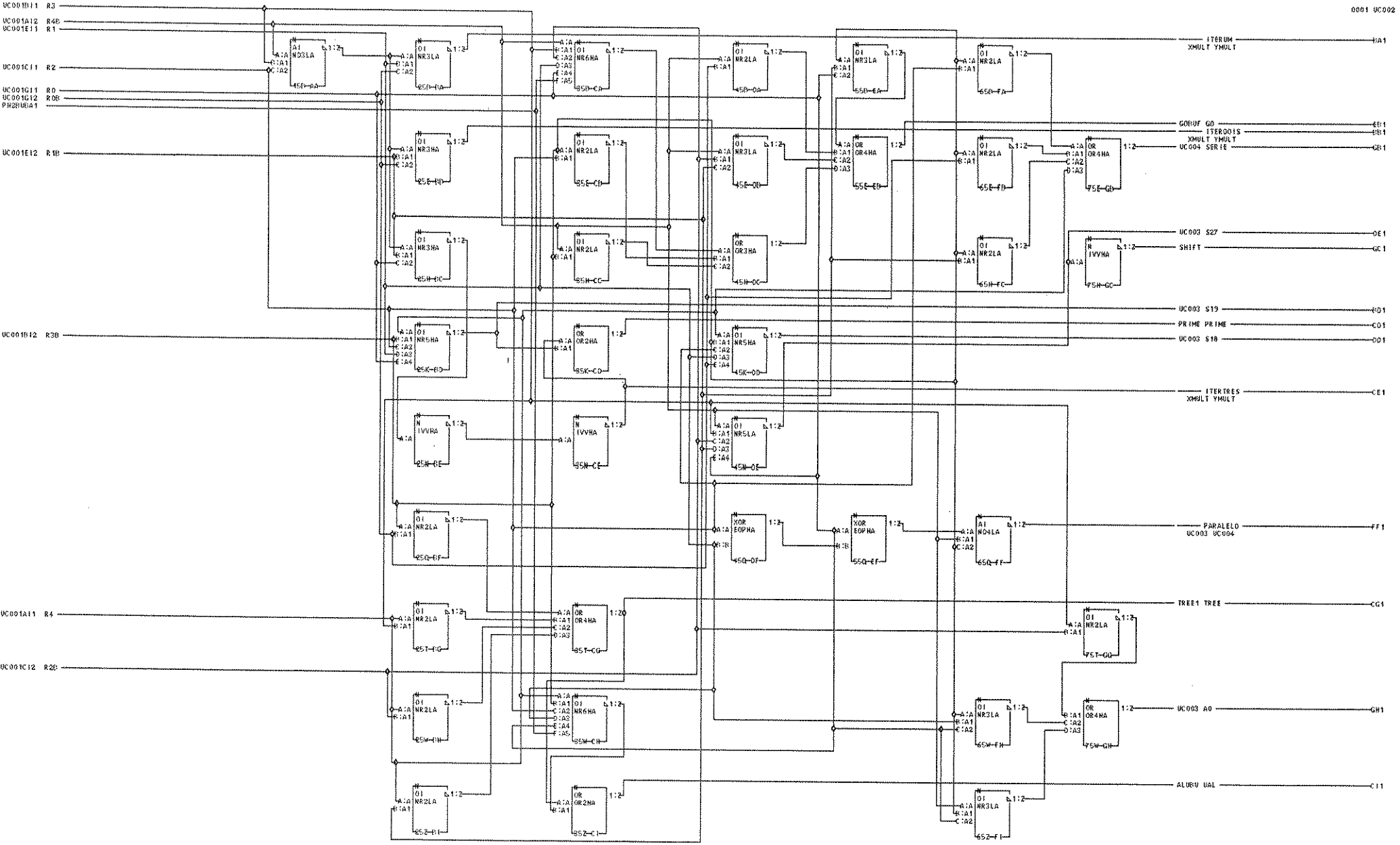
BUFFER PARA SINAL TREE

LOC=1A-A1A1A30A30AA01

USN 00001 PRI=02OCT91 1342

AUC= SEC
PFORM= NEXTBLK BC
MACH=RBSL
CID ZKOND JOB KONDORB


0001 UC001
R4B
UC002 UC003
R3B
UC002 UC003
R4
UC002 UC003
R1B
UC002 UC003
R2B
UC002 UC003
R3
UC002 UC003
UC002 UC003 UC004
R2
UC002 UC003
R1
UC002 UC003
R0
UC002 UC003 UC004
R0B
UC002 UC003
AFA 0A00
UC003GA1 RESET
PH1BUSA1
CONNECTORS
AFA
0001/1A-A1A1A30A30AA01
CONTADOR DE MODULO 30
LOC=1A-A1A1A30A30AA01
USN 00001
PRI=020CT91 1342
AUC=
SEC
PFORM=
NEXTBLK GJ
MACH=RRSL
CTO 2KGRD
JOB KONGORB
0001

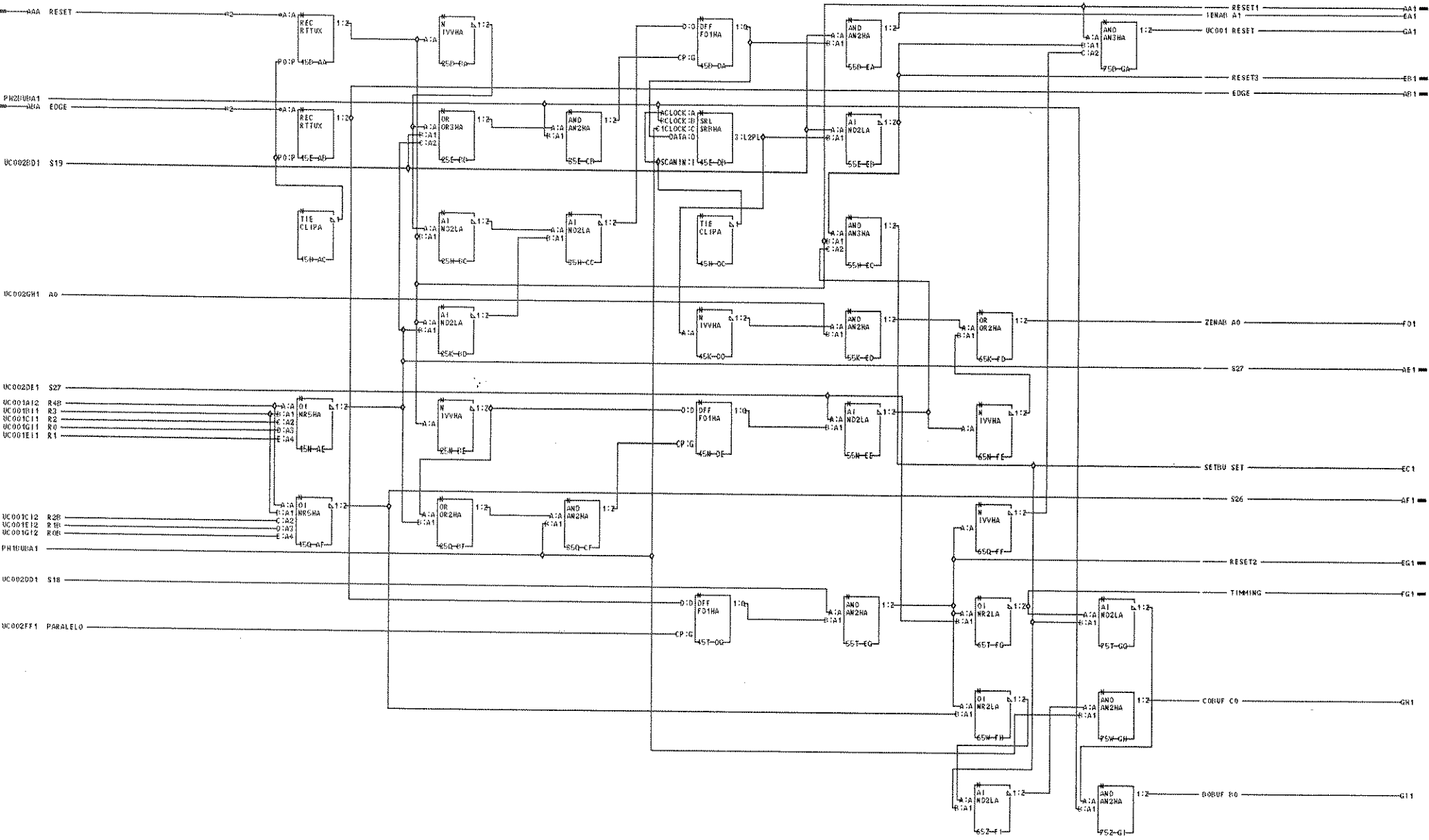


GERA SINAIS ITERUM, ITERODIS, ITERTRES, SERIE, PARALELO,GO, SHIFT, TREE, UAL, PRIME.

LOC=1A-A1A1A30A30AA01

USN 00001 PRT=020CT91 1342

AUC= SEC
PFORM= NEXTBLK GI
MACHERSL
CIS ZKOND JOB KONDORB



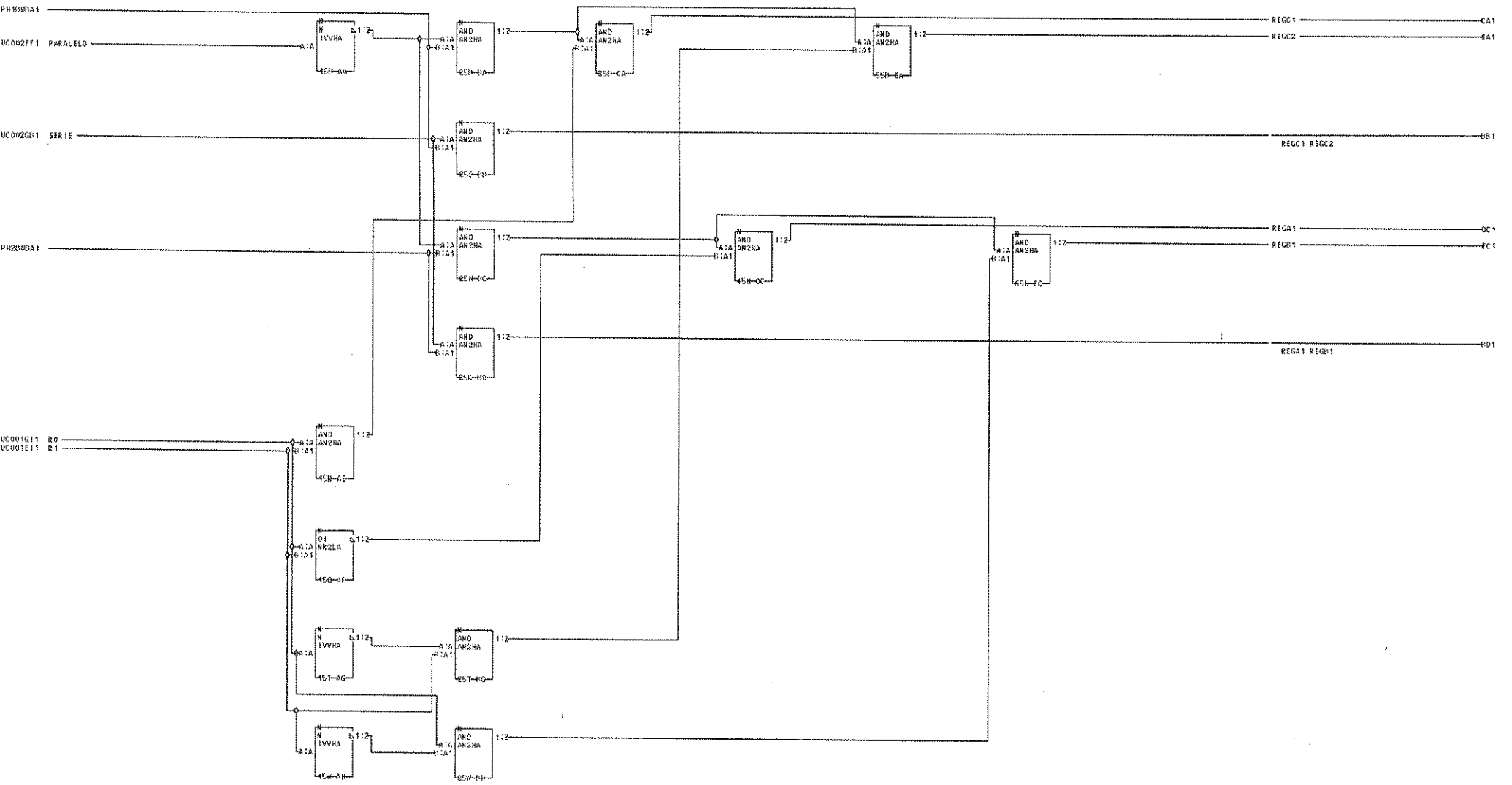
CONNECTORS
AAA
0001/1A-A1A1A30A30AA01
ABA
0002/1A-A1A1A30A30AA01

GERA SINAIS SETB, RESET, HABILIT A 1, HABILITA Z, TIMER.

LOC=1A-A1A1A30A30AA01

USN 00001 PRI=02OCT91 1342

AUC= SEC
PFORM= NEXTBLK GJ
MACH=PRSL
CIO ZKOND JOB KONGORB



DECODIFICADOR PARA REG ENTRADA

LOC:1A-A1A1A30A30AA01

USN 00001 PRI=020CT91 1342

AUC= SEC

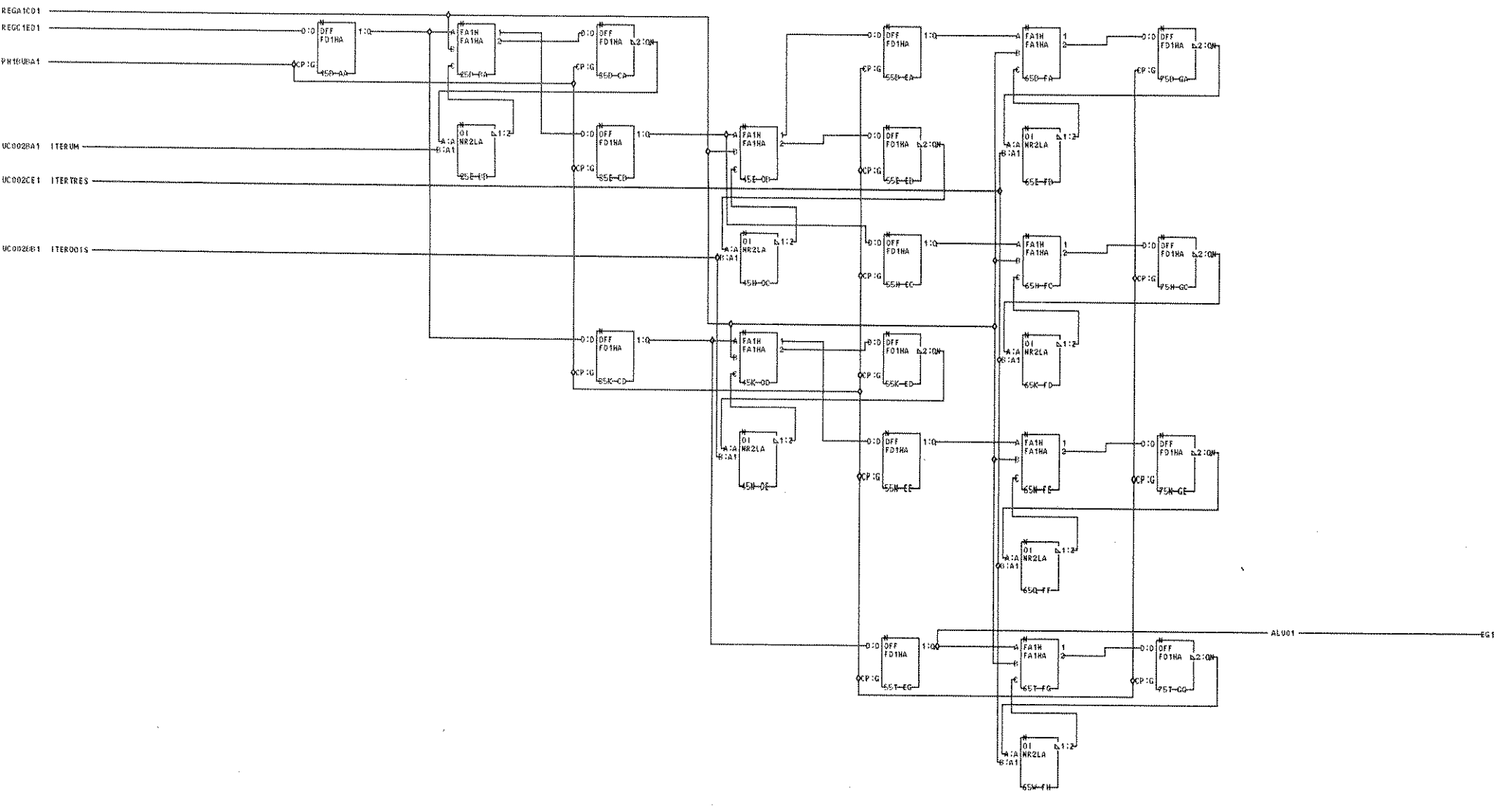
PFORM= NEXTBLK FD

MACH=ERSL

C10 ZKONG JOB KONCORE

UC004

0001



MULTIPLICADOR X

LOC=1A-A1A1A30A30AA01

USN 00001 PRI=020CT91 1342

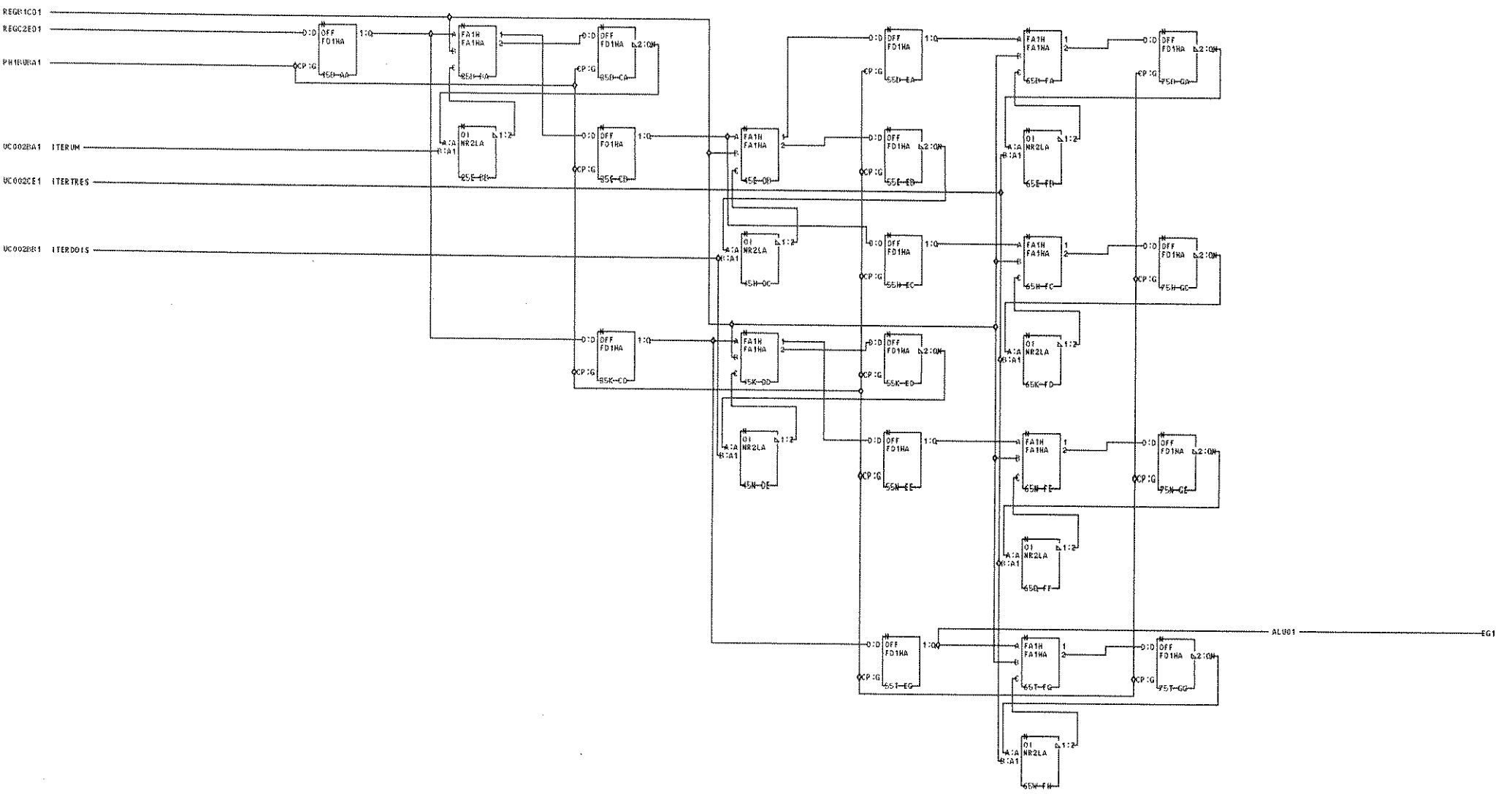
AUC= SEC

PFORM= NEXTBLK GR

MACHFRSL

CIO ZKONG JOB KONDORB

XMULT 0001



MULTIPLICADOR Y

LOC=1A-A1A1A30A30AA01

USN 00001 PRI=020CT91 1342

AUC= SEC

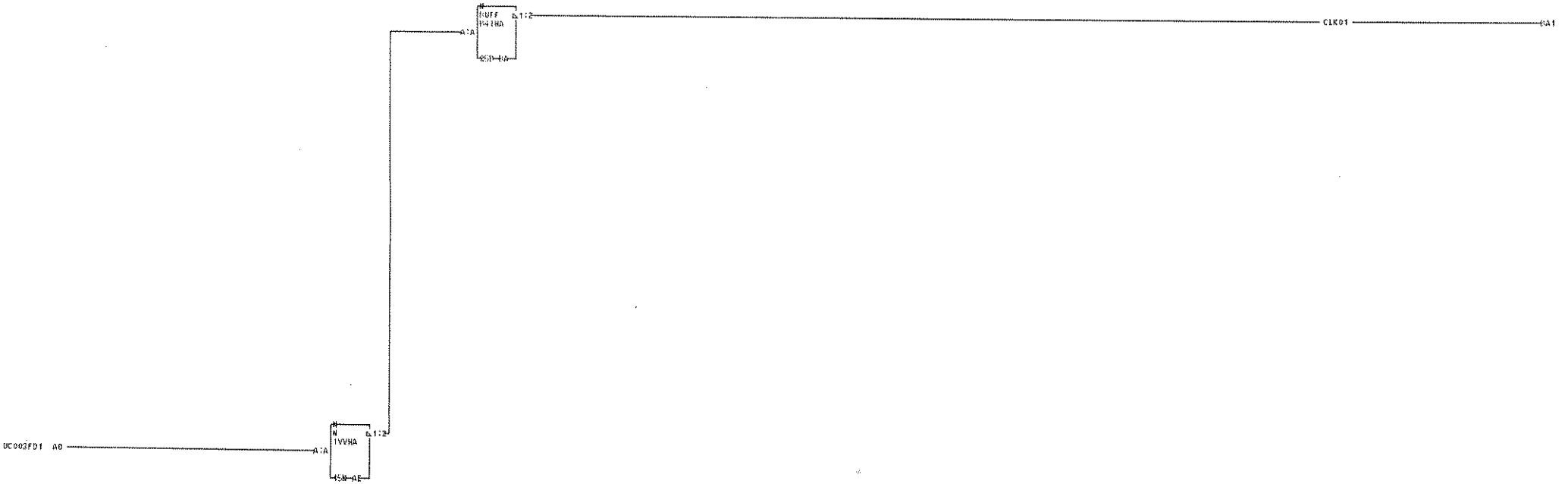
PFORM= NEXTBLK GH

MACH=PRSL

CID 2KOND JOB KONOORB

YMULT

0001



BUFFER DE HABILITA Z

LOC=1A-A1A1A30A30AA01

USN 00001 PR1=020C791 1342

AUC= SEC
PFORM= NEXTBLK 60
MACH=ERSL
CID ZKOND JOB KONDORB