COMPONENTES

No SAP-1 existe um único barramento principal, e apenas um componente de cada vez pode colocar dados nesse barramento. Se mais de um dispositivo tentar colocar valores diferentes ao mesmo tempo, haveria conflito e os dados ficariam corrompidos. Cada registrador e a memória possuem saídas do tipo três estados (tri-state outputs), permitindo que apenas a fonte de dados desejada seja habilitada a colocar seu valor no barramento enquanto todas as outras permanecem "desconectadas". Dessa forma, garante-se que o barramento transporte somente um dado por vez de forma coerente.

Contador de Programa: o programa é armazenado no começo da memória com a primeira instrução no endereço binário 0000. O CP conta de 0000 a 1111. Sua tarefa é enviar à memória o endereço da instrução seguinte a ser buscada e executada. O contador de programa é como alguém que aponta um dedo em uma lista de instruções, dizendo para fazer isto em primeiro lugar, fazer isto em segundo, fazer isto em terceiro etc.

REM (ou MAR): o REM retém o endereço da memória que se deseja acessar na RAM. Informamos à RAM qual posição (endereço) queremos ler para buscar a próxima instrução a ser executada.. Quando o PC fornece um endereço, esse sinal precisa ser estável durante a leitura. O PC, ao avançar, muda o endereço a cada pulso de clock. O REM é utilizado para "segurar" (latch) o endereço atual enquanto a RAM realiza a operação de leitura. Isso permite que o PC possa eventualmente ser modificado (incrementado para a próxima instrução) sem interromper a operação que já está em andamento na RAM. O REM serve como uma ponte entre o componente que gera o endereço e a RAM. Sem a MAR, se o PC alterasse o endereço rapidamente, a memória não teria tempo de "capturar" e usar o endereço certo, pois o barramento mudaria junto com o PC. A MAR garante que, mesmo se o PC já estiver incrementando para o próximo endereço, o valor do endereço anterior fica guardado e estável para a memória fazer sua leitura com calma. Você pode incrementar o PC internamente sem mostrar nada no barramento. Só quando quiser que a memória use esse endereço, você deixa o PC colocar seu valor no barramento. Então a MAR "pega" esse valor do barramento, garantindo que o endereço fique firme e não desapareça se o PC mudar novamente. Isso evita que o dado a ser lido da memória se perca ou fique instável.

RAM: Antes de iniciar a execução, você "carrega" a RAM com as instruções do programa e dados necessários. Durante a execução, a RAM recebe um endereço e, a partir dele, retorna o conteúdo armazenado nesse end. (instrução ou dado) para o barramento de dados (barramento W no SAP-1). Armazena o conjunto de instruções e dados que formam o programa.

Registrador de instruções (IR): armazena a instrução atualmente em execução. Uma vez que a RAM coloca a instrução no barramento, precisamos guardá-la em um lugar estável para que o c/s possa analisá-la e gerar os sinais de controle adequados. A instrução precisa estar estável, disponível a qualquer momento ao longo do ciclo de execução, não apenas no instante que sai da memória. Se dependêssemos da RAM diretamente, assim que mudássemos de endereço ou passássemos para outra etapa, perderíamos a informação da instrução anterior. Ter o IR permite que a instrução fique acessível por todo o período de sua execução, sem depender mais da memória. O IR divide a instrução em partes (o nibble superior para o código da operação e o nibble inferior para o operando). Mantém a instrução atual disponível para que o c/s possa decodificá-la e gerar sinais de controle adequados. Sem o IR, perderíamos a referência da instrução assim que o barramento mudasse ou a RAM fosse acessada para outra operação.

Controlador-Sequencializador: Os 12 bits que vêm para fora do controlador- sequencializador formam uma palavra que controla o resto do computador (como um supervisor que diz aos outros o que fazer). Os 12 fios que transportam a palavra de controle são chamados barramento de controle. A palavra de controle tem o formato de:

$$\mathbf{CON=}\ C_P E_P \overline{L_M}\ \overline{CE}\ \ \overline{L_I}\overline{E_I}\overline{L_A} E_A\ \ S_U E_U \overline{L_B}\overline{L_O}$$

Esta palavra determina como os registradores reagirão à próxima transição positiva de relógio. O c/s, que é como o "diretor de orquestra", começa em um estado de reset inicial quando o SAP acaba de ser ligado ou sofre um reset. Esse estado inicial define quais sinais de controle estarão ativos já no primeiro ciclo de clock. Assim, quando o primeiro pulso de clock acontece, o computador não está “em branco”: o c/s já está num estado pré-definido que faz parte do projeto. Esse estado inicial do controlador faz com que, na primeira “batida” do relógio, ele ative os sinais necessários para iniciar a primeira operação de busca de instrução.

Cada passo contido no c/s é desenhado pelo projetista da máquina. No estado inicial (digamos “T0”), o c/s sabe que precisa buscar a primeira instrução da memória.O que isso significa? Significa que foram projetados sinais de controle para este estado inicial, tais como:

-Habilitar a saída do PC (Ep alto, por exemplo) para colocar o endereço 0 no barramento.

-Permitir que a MAR (REM) carregue o endereço do barramento (Lm baixo, por exemplo).
Estes sinais são definidos pelo projeto lógico do c/s, que não é “inteligente”, mas segue uma lógica combinacional e sequencial embutida nele. Ao ligar o computador, o c/s entra no estado em que esses sinais já são configurados dessa forma.

Acumulador (A): registrador de memória intermediária que armazena respostas intermediárias durante um processamento no computador. O acumulador tem 2 saídas. A saída de 2 estados vai diretamente ao somador-subtrator. A saída de três estados vai ao barramento W. Portanto, a palavra do acumulador de 8 bits continuamente comanda o somador-subtrator; a mesma palavra aparece no barramento W quando EA está alto.

Somador-subtrator: o SAP-1 usa um somador-subtrator de complemento de 2. Quando SU for baixo, a soma fora do somador-subtrator será S=A+B. Quando SU for alto, aparecerá a diferença: S = A + B' (B’ é o número B negativo). O somador-subtrator é assíncrono (não-sincronizado); isto significa que seu conteúdo pode variar logo que as palavras de entrada variem. Quando EU for alto, estes conteúdos aparecerão no barramento W.

Registrador B: Usado em operações aritméticas. Um baixo LB e uma transição positiva de relógio carregam a palavra do barramento W dentro do registrador B. A saída de dois estados do registrador B comanda o somador-subtrator, fornecendo o número a ser adicionado ou subtraído do conteúdo do acumulador.

Registrador de saída: No final de um processamento do computador, o acumulador contém a resposta ao problema que está sendo resolvido. Neste ponto, necessitamos transferir a resposta para o mundo exterior. Isto é onde é usado o registrador de saída. Quando EA for alto e L0 for baixo, a próxima transição positiva de relógio carregará a palavra do acumulador no registrador de saída.

++

OPERAÇÕES

LDA: Load the Accumulator. Uma instrução LDA completa inclui o endereço hexadecimal dos dados a serem carregados. LDA 8H, por exemplo, significa "carregar o acumulador com o conteúdo do local 8H da memória". Similarmente, LDA AH significa "carregar o acumulador com o conteúdo do local AH da memória", LDA FH significa "carregar o acumulador com o conteúdo do local FH da memória " etc.

ADD: Uma instrução ADD completa inclui o endereço da palavra a ser acrescentada. Por exemplo, ADD 9H significa "acrescentar o conteúdo do local 9H " da memória ao conteúdo do acumulador"; a soma substitui o conteúdo original do acumulador. A palavra RAM endereçada vai ao registrador B e a saída do somador-subtrator vai para o acumulador.

SUB: Uma instrução SUB completa inclui o endereço da palavra a ser subtraída. Por exemplo, SUB CH significa "subtrair o conteúdo do local CH da memória do conteúdo do acumulador"; a diferença para fora do somador-subtrator depois substituir o conteúdo original do acumulador.

OUT: A instrução OUT diz ao computador SAP-1 para transferir o conteúdo do acumulador para a porta de saída. Depois de OUT ter sido executada, podemos ver a resposta ao problema que está sendo resolvido. OUT é completa por si própria; isto é, não temos que incluir um endereço quando usamos OUT porque a instrução não envolve dados na memória.

HLT: Esta instrução diz ao computador para parar o processamento de dados. HLT assinala o fim de um programa, similar à maneira com que um ponto assinala o fim de uma frase ou sentença. Devemos usar uma instrução HLT no fim de cada programa do SAP-1; do contrário, obtemos refugos (trash) no computador (respostas sem significado causadas pelo processamento descontrolado). HLT é completa por si própria; não temos que incluir uma palavra de RAM ao usarmos HLT porque esta instrução não envolve a memória.

Mnemônicos	Operação
LDA	Carregue os dados da RAM no acumulador.
ADD	Some os dados da RAM com o acumulador.
SUB	Subtraia os dados da RAM do acumulador .
OUT	Carregue os dados do acumulador no registrador de saída.
HLT	Pare o processamento.

++

PROGRAMAR O SAP-1

Quando programando as chaves de dados com uma instrução, o código de operações entra no nibble superior, e o operando (o resto da instrução) entra no nibble inferior.

Mnemônicos	Código op
LDA	0000
ADD	0001
SUB	0010
OUT	1110
HALT	1111

Para armazenar as instruções, convertemos cada instrução em binário.

A linguagem de montagem (Assembly language) envolve o trabalho com mnemônicos quando se escreve um programa. A linguagem de máquina envolve o trabalho com cordões de 0s e 1s.

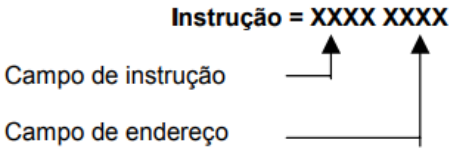
Endereço	Instrução
0H	LDA 9H
1H	ADD AH
2H	ADD BH
3H	SUB CH
4H	OUT
5H	HLT

Este programa está em linguagem de montagem conforme ele se acha agora. Para obtê-lo em linguagem de máquina, nós o traduzimos em 0s e 1s conforme segue:

Endereço	Instrução
0000	00001001
0001	00011010
0010	00011011
0011	00101100
0100	1110 XXXX
0101	1111 XXXX

Qualquer programa como o precedente que esteja escrito em linguagem de máquina é chamado programa objeto. O programa original com mnemônicos é chamado programa fonte. Em SAP-1, o operador traduz o

programa fonte em um programa objeto quando da programação das chaves de dados e de endereços. Um detalhe final. Os quatro MSBs de uma instrução em linguagem de máquina SAP-1 especificam a operação, e os quatro LSBs dão o endereço. As vezes nós nos referimos aos MSBs como o campo de instrução e aos LSBs como o campo de endereço. Simbolicamente,



++

Passo 1: O PC fornece o endereço da próxima instrução. Esse endereço é carregado no REM, garantindo que a RAM tenha o endereço estável.

Passo 2: A RAM, ao receber o endereço do REM, coloca a instrução referente àquele endereço no barramento W.

Passo 3: A instrução no barramento W é carregada no IR.
Agora o IR retém a instrução completa, dividindo-a em parte de código de operação (opcode) e parte de dado/endereço (operando).

Passo 4: O c/s lê o opcode do IR para gerar os sinais de controle apropriados.

Passo 5: Enquanto isso, o PC pode ser incrementado, preparando-se para a próxima instrução. O REM pode receber um novo endereço no próximo ciclo, a RAM pode ser novamente acessada, e assim por diante.

O Primeiro Pulso de Clock
Assim que damos o primeiro pulso de clock (a primeira borda de subida), o que acontece?

- O PC, que já estava com valor 0, é habilitado a colocar este 0 no barramento (por causa do sinal Ep). Isso não requer nenhuma "inteligência" extra: o PC apenas obedece. Quando Ep (Enable PC) está alto, a saída do PC é conectada ao barramento.
- Ao mesmo tempo, o sinal Lm (Load MAR) está baixo, o que significa que na próxima transição do clock a MAR irá carregar o valor presente no barramento. Então, na borda do clock, a MAR pega o endereço 0 que está no barramento e o armazena.

Em outras palavras, no primeiro clock, não é que o SAP "descobre sozinho" o que fazer. Ele faz o que foi projetado para fazer quando recém iniciado: o c/s está num estado inicial bem definido que determina que o primeiro passo é pegar o valor do PC e colocá-lo na MAR.

A Sequência Continua
Agora, após o primeiro pulso, a MAR tem o endereço 0. No próximo passo (o segundo ciclo de clock), o c/s muda de estado (ele avança para o próximo estágio do ciclo de busca). Neste segundo estágio, outro conjunto de sinais de controle será ativado. Por exemplo:

- Pode-se habilitar a RAM a colocar o conteúdo da memória posição 0 no barramento.
- Simultaneamente, o IR (registrador de instruções) pode receber o comando para carregar o dado do barramento.

Cada etapa do ciclo de busca e execução é projetada para acontecer em sequência, e o c/s é uma máquina de estados que transita de um estado para o próximo a cada pulso de clock. Em cada estado, sinais de controle bem definidos são emitidos, guiando o fluxo de dados

Por que há necessidade de manter dados estáveis com o REM e o IR?

O SAP-1, assim como a maioria dos computadores, funciona a partir de um ciclo de busca e execução de instruções. Esse ciclo é dividido em etapas sincronizadas por pulsos de clock. Em cada etapa, certos sinais mudam de estado e certos componentes devem manter valores estáveis por tempo suficiente para serem usados em outra etapa.

O REM (registrador de endereço da memória) é necessário porque a memória (RAM) precisa receber um endereço estável durante todo o período de leitura (ou escrita). Quando dizemos "estável", isso significa que, enquanto a RAM está sendo acessada, o endereço não pode ficar mudando.
Imagine o seguinte:

1- O PC (contador de programa) gera um endereço. 2- Esse endereço é colocado no REM. 3- Enquanto a RAM lê a posição indicada pelo REM, o PC pode estar sendo incrementado internamente para a próxima instrução. Se não existisse o REM, assim que o PC mudasse o endereço, a RAM poderia "ver" esse novo endereço antes de **terminar a leitura do anterior, causando resultados imprevisíveis.**

O IR (registrador de instruções) guarda a instrução atual para que o c/s possa decodificá-la e gerar os sinais de controle necessários. A instrução não é algo que você usa apenas no momento exato em que a RAM a entregou. Uma instrução típica no SAP-1 precisa ser decodificada e executada em vários passos (por exemplo: no primeiro passo busca a instrução, no segundo passo decodifica o opcode, no terceiro executa a operação...). Durante esse tempo, outros componentes do computador vão mudar seu estado. A RAM provavelmente estará sendo acessada para outras coisas, o PC estará sendo incrementado, etc. Sem o IR, perderíamos a referência clara de qual instrução estamos executando assim que a memória mudasse para exibir a próxima.

A "instabilidade" aqui não é um fenômeno mágico. É simplesmente o fato de que, assim que o endereço da memória muda para buscar a próxima instrução, o conteúdo do barramento W também muda. Se não guardamos a instrução em algum lugar (o IR), ela deixa de estar disponível para o controlador. O IR oferece "tempo" para o c/s analisar o opcode e gerar os sinais de controle ao longo de diversos estados do ciclo de clock, mantendo a instrução atual presente durante todo o processo de execução.

Em resumo, a "necessidade de tempo" e "instabilidade" não é algo vago: é consequência da sequência de etapas em que o computador trabalha. Cada pulso de clock inicia uma etapa nova. Se os valores não forem segurados em registradores específicos, eles "somem" ou mudam antes de serem usados completamente.

Como o controlador/sequencializador (c/s) gera os sinais a partir da instrução?

No SAP-1, o formato de uma instrução típica é um byte: O nibble (4 bits) superior representa o código da operação (OPCODE). O nibble inferior representa o operando (por exemplo, um endereço de memória ou um dado imediato).

O IR recebe essa instrução completa. O nibble superior (OPCODE) é encaminhado diretamente para o c/s. O c/s é um circuito lógico (basicamente uma máquina de estado finito) que usa o OPCODE como entrada, juntamente com o estado interno do sequenciador (por exemplo, qual passo da execução estamos) para determinar quais sinais de controle ativar.

Porque o c/s precisa do endereço que vem do IR?

Imagine o controlador/sequencializador (c/s) como um maestro extremamente talentoso, mas que precisa de uma “partitura” para saber exatamente qual música conduzir. O c/s sabe muito bem **como** orquestrar cada passo da execução de uma instrução (quando habilitar o PC, quando carregar a MAR, quando ler da RAM, etc.), porém não sabe **qual** instrução o computador precisa executar a seguir. Quem fornece essa informação é o IR (Registrador de Instruções).

Pense assim:

- O c/s conhece todos os "ritmos" (micro-operações) e sabe a sequência certa para executá-los, mas não sabe qual música deve tocar agora.
- O IR contém a instrução atual (o "nome da música"), isto é, o código de operação (opcode) que diz ao c/s:
    - Se é para somar dois valores,
    - Se é para carregar um dado da memória,
    - Se é para pular para outra instrução,
    - Entre outras operações.

Sem o IR, o c/s apenas saberia repetir o mesmo ciclo básico (buscar, decodificar, executar) de forma genérica, mas não teria ideia de qual tipo específico de operação realizar. Ao receber o opcode do IR, o c/s escolhe qual conjunto de sinais de controle ativar para que a ação apropriada àquela instrução aconteça.

1. O Papel do OPCODE

O OPCODE, contido no IR, informa **qual** instrução está sendo executada. Por exemplo, um OPCODE pode indicar uma instrução de soma, outra pode ser de carregamento de dados da memória, outra pode ser de armazenamento, etc.

Se pensarmos em termos de um “guia de referência rápida”, o OPCODE é como uma "chave" que seleciona uma certa sequência de ações necessárias para executar aquela instrução.

2. Por que o OPCODE não basta sozinho?

Cada instrução, mesmo sendo identificada por um único OPCODE, não acontece "instantaneamente". Ela precisa ser desdobrada em um conjunto de passos elementares (micro-operações). Por exemplo, para executar uma instrução de soma, o computador pode precisar:

- No primeiro passo: Buscar a instrução da memória (fetch)
- No segundo passo: Decodificar a instrução (saber que é soma)
- No terceiro passo: Ler o dado a ser somado de um registrador ou da memória
- No quarto passo: Acionar a unidade de soma (ULA)
- No quinto passo: Armazenar o resultado no registrador de destino

Cada instrução terá o seu conjunto específico de passos, mas quase sempre passará por uma etapa de busca e decodificação antes da execução do OPCODE em si.

3. O Ciclo Interno do Controlador/Sequencializador (c/s)

O c/s tem um mecanismo para acompanhar essas etapas. Esse mecanismo é muitas vezes um contador interno de estados (por exemplo, estados T0, T1, T2, T3...) que avança a cada pulso de clock. Essa contagem de "tempo interno" não é o mesmo que o PC (contador de programa); é um contador ou gerador de fases internas do ciclo de instrução.

Em outras palavras, o c/s combina:

- **O estado interno de tempo** (em que passo do ciclo de busca/execução estamos?), e
- **O OPCODE** (qual instrução está sendo executada?)

Com essas duas informações, o c/s determina quais sinais de controle ativar em cada etapa. Pense nisso como uma matriz:

- O eixo horizontal da matriz pode ser o OPCODE (qual instrução).
- O eixo vertical da matriz são as etapas temporais (T0, T1, T2...), ou seja, o progresso do ciclo.

Em cada combinação (OPCODE + Estado de Tempo), o c/s gera uma "palavra de controle" específica, ativando e desativando as linhas de controle adequadas.

4. Por que esse "ciclo interno" é necessário?

O OPCODE diz “o que fazer”, mas não diz “quando fazer”. As instruções precisam ser decompostas em passos sequenciais. Cada passo ocorre em um ou mais ciclos de clock. O c/s precisa de uma referência temporal (o estado interno) para saber qual parte da instrução está em andamento. Sem esse ciclo interno, o c/s teria apenas o OPCODE, mas não saberia se está no momento de buscar a instrução, decodificá-la, executar a operação aritmética ou armazenar o resultado.

Assim, o ciclo interno do c/s é essencial para:

- Estabelecer a ordem correta das micro-operações.
- Garantir que primeiro se busque a instrução da memória, depois se decodifique, e só então se execute a operação especificada pelo OPCODE.

5. Resumindo

- **OPCODE**: Informa qual é a instrução, ou seja, qual sequência de micro-operações deve ser realizada.
- **Estado interno do c/s (ciclo interno)**: Informa em qual etapa dessa sequência estamos. Isso permite que as micro-operações sejam divididas no tempo, uma após a outra, ao invés de tentar fazer tudo ao mesmo tempo.

Sem o estado interno do c/s, você até saberia **qual** instrução executar (pelo OPCODE), mas não teria o “roteiro temporal” para dividir essa execução em etapas lógicas e ordenadas. Com o estado interno do c/s, cada instrução é “coreografada” passo a passo, na ordem correta e no tempo certo.

- O c/s possui um "roteiro" interno, uma espécie de sequência fixa de etapas (os estados T0, T1, T2, etc.) que avançam a cada pulso de clock.
- O OPCODE funciona como um índice que seleciona qual “roteiro” (ou qual conjunto de micro-operações) o c/s deve seguir.
- Assim, o c/s não apenas sabe em que momento do ciclo da instrução está, mas também qual série de passos deve executar com base no OPCODE fornecido pelo IR.

Dessa forma, o estado interno do c/s (os passos) combinado com o OPCODE (a instrução) permite que o c/s gere a sequência de sinais de controle correta para cada instrução, na ordem e no tempo certos. Isso garante que cada instrução seja executada adequadamente, etapa por eta