GABARITO – AD1 – ORGANIZAÇÃO DE COMPUTADORES

1. (1,0) Considere uma máquina que possa endereçar 512 Mbytes de memória física, utilizando endereço referenciando byte, e que tenha a sua memória organizada em blocos de 16 bytes. Ela possui uma memória cache que pode armazenar 8K blocos, sendo um bloco por linha. Mostre o formato da memória cache, indicando os campos necessários (válido, tag, bloco) e o número de bits para cada campo, e o formato de um endereço da memória principal, indicando os bits que referenciam os campos da cache, para os seguintes mapeamentos:

a) Mapeamento direto.

M*emória Principal*

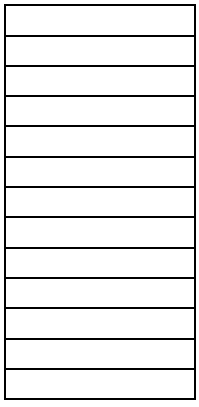
- *Tamanho da memória (em bytes) = 512Mbytes, como cada célula contém 1 byte, temos, então, N = 512M células*
- *A MP está organizada em blocos de 16 bytes, como cada célula = 1 byte (K = 16 células/bloco)*
  N = 512M células e K = 16 células / bloco, o total de blocos da MP (B) será:
  Total de blocos: B = N / K => B = 512M células / 16 células/bloco => B = 32 M blocos

Memória Cache

O*BS: O K (quantidade de células/bloco) tem de ser igual a MP.*

- *Tamanho da memória cache (em blocos ou linhas) => Q = 8K blocos*
- *Tamanho da memória cache em células = Q x K = 8K blocos x 16 células/blocos* = 128 K células (128 Kbytes)

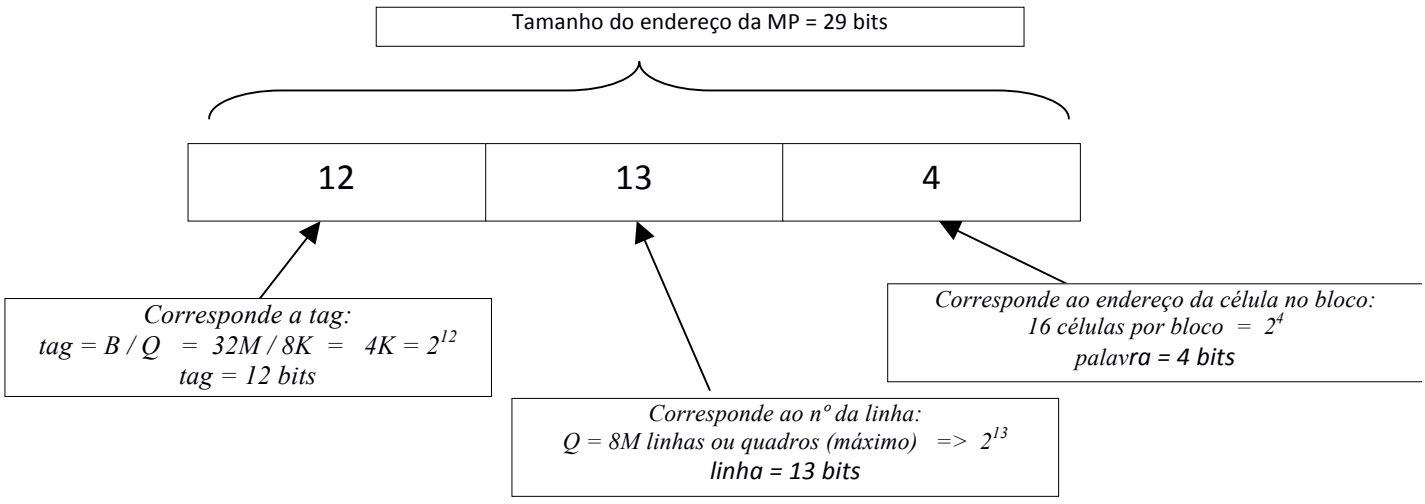
Memória principal
512M células: N
32M blocos: B



Organização da cache

linha	válido	tag	Conteúdo (bloco)
0	1 bit	12 bits	16 células de 8 bits cada = 128bits
1			
2			
3			
4			
5			
…..			
Q - 2			
Q - 1			

Para endereçarmos toda a MP precisamos da seguinte quantidade de bits (E)
sendo $N = 2^E$ => N = 512M células => $N = 2^{29}$ => E = 29 bits

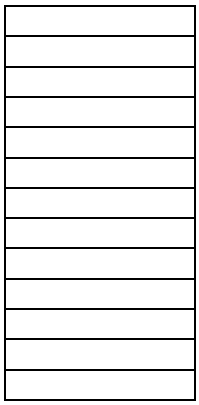


b) **Mapeamento totalmente associativo.**

Memória Principal
=> N = 512M células
=> K = 16
=> B = 32 M blocos

Memória Cache
OBS: O K (quantidade de células/bloco) tem de ser igual a MP.
=> Q = 8K blocos
=> Tamanho da memória cache = 128Kbytes

Memória principal
512M células: N
32M blocos: B

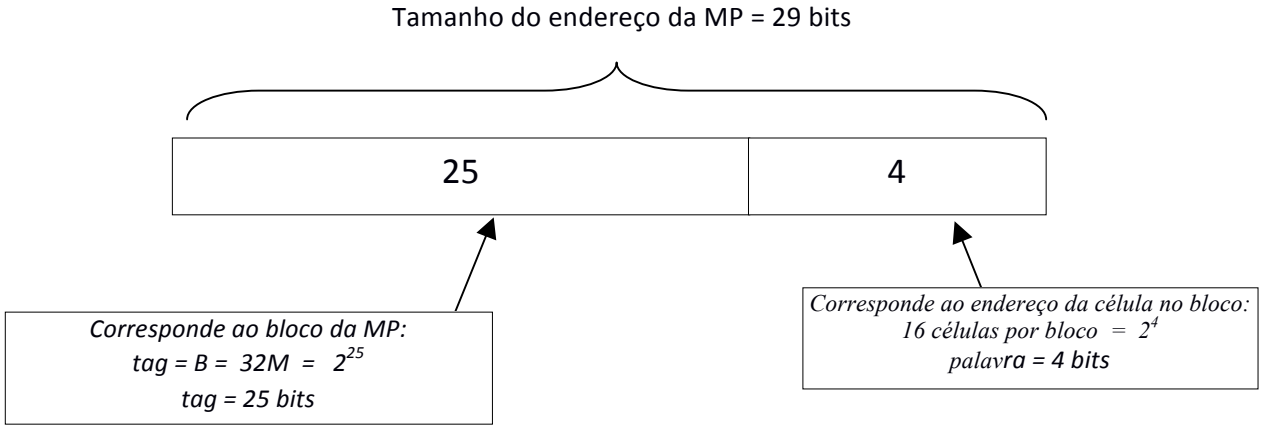


Organização da cache

linha	válido	tag	Conteúdo (bloco)
0	1 bit	25 bits	16 células de 8 bits cada = 128bits
1			
2			
3			
4			
5			
.....			
Q - 2			
Q - 1			

Para endereçarmos toda a MP precisamos da seguinte quantidade de bits: E = 29 bits

Como o bloco pode ser alocado em qualquer posição da memória cache. A tag indicará qual dos blocos da MP está alocado naquela posição da memória cache



c) Mapeamento associativo por conjunto, onde cada conjunto possui quatro linhas, cada uma de um bloco.

Memória Principal

=> K = 16 (quantidade de células/bloco)

=> B = 32M blocos

***=>** N = 512M células*

Memória Cache

O*BS: O K (quantidade de células/bloco) tem de ser igual ao da MP.*

=> Q = 8K blocos

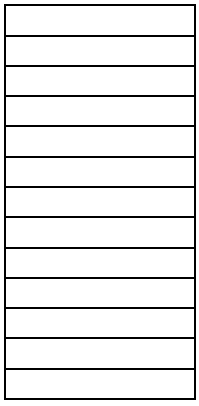
=> Tamanho da memória cache = 128Kbytes

=> 1 conjunto = 4 linhas (ou quadros) => Total de conjuntos (C) = 8K células / 4 => C = 2K conjuntos

Memória principal

512M células: N

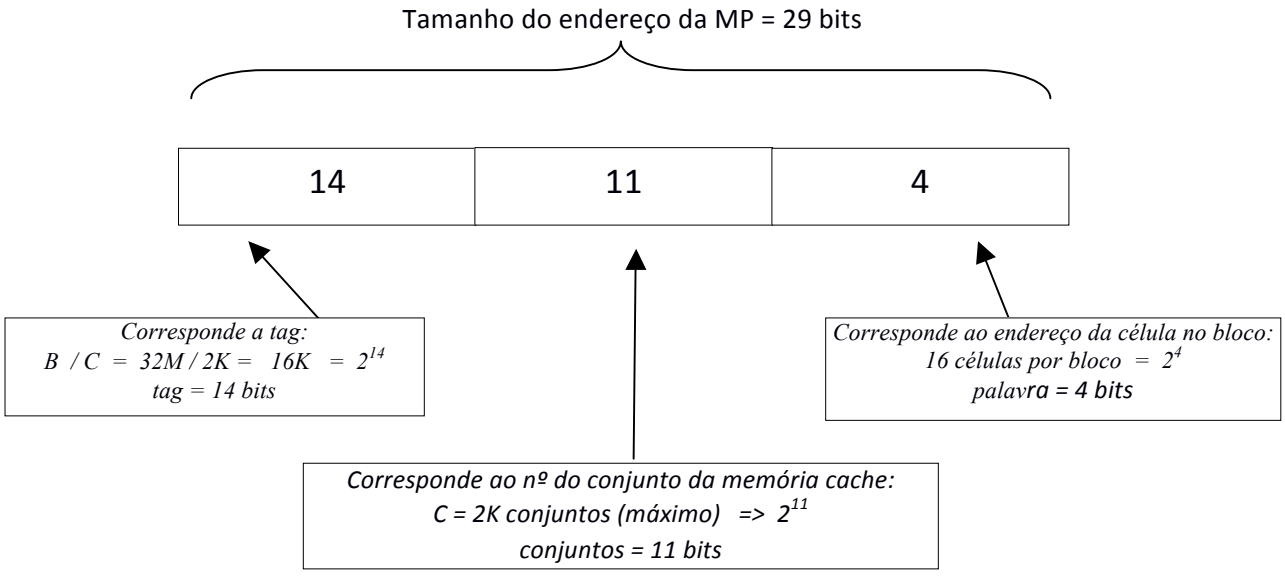
32M blocos: B



Organização da cache

Conjunto	linha	válido	tag	Conteúdo (bloco)
	0	1 bit	14 bits	16 células de 8 bits cada = 128bits
	1			
0	2			
	3			
	4			
1	5			
		…..		
C - 1	Q - 2			
	Q - 1			

P*ara endereçarmos toda a MP precisamos da seguinte quantidade de bits: E = 29 bits*



2. (1,0) Explique em detalhes a organização hierárquica do subsistema de memória nos computadores atuais.

Organização hierárquica dos subsistemas de memória:

O subsistema de memória é interligado de forma bem estruturada e organizado hierarquicamente em uma pirâmide com os níveis descritos a seguir.

No topo da pirâmide teríamos os registradores, que são pequenas unidades de memória que armazenam dados na UCP. São dispositivos de maior velocidade com tempo de acesso em torno de 1 ciclo de memória, menor capacidade de armazenamento além de armazenar as informações por muito pouco tempo.

Em um nível abaixo teríamos a memória cache, cuja função é acelerar a velocidade de transferência das informações entre UCP e MP e, com isso, aumentar o desempenho do sistema. A UCP procura informações primeiro na Cache. Caso não as encontre, as mesmas são transferidas da MP para a Cache. A cache possui tempo de acesso menor que a da Memória principal, porém com capacidade inferior a esta, mas superior ao dos registradores e o suficiente para armazenar uma apreciável quantidade de informações, sendo o tempo de permanência do dado menor do que o tempo de duração do programa a que pertence.

Abaixo da memória cache teríamos a memória básica de um sistema de computação, que é a memória principal. Dispositivo onde o programa (e seus dados) que vai ser executado é armazenado para que a UCP busque instrução por instrução para executá-las. A MP são mais lentas que a cache e mais rápidas que a memória secundária, possui capacidade bem superior ao da cache e os dados ou instruções permanecem na MP enquanto durar a execução do programa.

3. (1,0) Faça uma pesquisa e explique a organização de memórias no processadores Multicore do Intel i7.

O processador I7 como o I3 e I5 utilizam a microarquitetura Sandy Bridge. Esta arquitetura tem como principais características:

- *A ponte norte, que controla memória, vídeo e controlador do barramento PCI Express, está integrada no mesmo chip do processor.*
- *Possui arquitetura em Anel, onde os componentes internos do processador se comunicam. Quando o componente quer "conversar" com outro, ele coloca a informação no anel para que ela chegue até o destinatário.*
- *Possui uma cache de instruções decodificadas denominada cache L0, capaz de armazenar cerca de 1536 instruções, em torno de 6KB.*
- *A cache L1 é dividida em 2: cache de instruções com cerca de 32KB, e cache de dados de igual tamanho. A cache L2 foi renomeado para cache intermediário com 256KB por núcleo. A cache L3 também foi renomeada, esta para cache de último nível, não mais unificada, e é compartilhada entre os núcleos, ou seja, não é ligada a um núcleo em particular. Qualquer núcleo pode usar qualquer um dos caches L3. As caches L3 podem ser utilizadas pelo componente gráfico para armazenar dados, em especial texturas aumentando o desempenho 3D, sem a necessidade de buscar dados na RAM.*
- *Para acesso a memória principal, possui controlador de memória DDR3 de dois canais, suportando memórias de até DDR3-1333*
- *Introdução de um conjunto de instruções AVX (Advanced Vector Extensions ou Extensões de Vetor Avançadas), estas instruções utilizam o conceito SIMD, armazenando dados vetorialmente e processá-los em uma única instrução de processamento. Este conjunto compõem-se de cerca de 12 novas instruções.*
- *Possui processador de vídeo integrado. A microarquitetura Sandy Bridge permite até 12 unidades de execução gráficas.*
- *Possui uma nova versão da tecnologia Turbo Boost que faz overclock no processador quando este demanda mais poder de processamento, e na arquitetura Sandy Bridge, esta tecnologia permite exceder seu TDP (termal Design Power) por até 25 segundos.*

Organização das memórias: ((fonte: www.clubedohardawre.com.br)

1) Registradores:

=> registradores da arquitetura 32 bits

- *Os registradores de uso geral (16 bits) são AX, BX, CX, DX*
- *O Intel x86 possui ainda registradores de 16 bits para segmentos de memória CS, DS, SS, ES, FS, GS.*
- *Os registradores de uso geral (32 bits) são EAX, EBX, ECX, EDX*

- *Os registradores apontadores de memória (32 bits) são ESI, EDI , EBP e ESP (este ponteiro de pilha)*
- *IPR - (Instruction Pointer Register) é o registrador de 32 bits como apontador de posição de memória (CI)*
- *RFLAGS (32 bits) flags*
- *Registradores para ponto flutuante ST0–ST7 (32 bits),*
- *vetores SIMD (Single Instruction, Multiple Data) MM0–MM7 (64 bits) e XMM0-XMM7 (128 bits).*
- *Registradores MMX0-7 de 64 bits*

=> registradores da arquitetura 64 bits

Além dos registradores dos 32 bits (manter compatibilidade), são acrescidos:

- *Instruções AVX, implementadas a partir da arquitetura Sandy Bridge, utilizam registradores XMM de 256 bits com o objetivo de armazenar vários dados menores e processá-los em uma única instrução (conceito SIMD)*
- *Registradores de 64 bits (RAX, RBX, RCX, RDX, RSI, RDI , RBP e RSP)*
- *Registradores de 80 bits de ponto flutuante (FPR0-7)*
- *IPR (Instruction Pointer Register) passa a ser de 64 bits (apontador de posição de memória)*
- *RFLAGS (64 bits) flags*

2) Organização da memória cache:

=> Série Ix (I3, I5, I7)

A tecnologia Intel de processadores Intel Sandy Bridge foi chamada de segunda geração da familía Intel Core, tem como característica a expessura de 32nm. A cache dessa arquitetura segue a seguinte organização:

- *Cache de microinstruções decodificadas (cache L0, capaz de armazenar 1.536 microinstruções, o que equivale a mais ou menos 6 kB);*
- *Cache L1 de instruções de 32 kB e cache L1 de dados de 32 kB por núcleo (nenhuma mudança em relação à arquitetura Nehalem – arquitetura anterior);*
- *O cache de memória L2 - cache intermediário com 256 kB por núcleo;*
- *O cache L3 é chamado é compartilhado entre os núcleos do processador e o processador gráfico, e variam a capacidade conforme de acordo com o processador:*
    - *Os processadores I7 estão disponíveis nas versões de quatro, seis núcleos e oito núcleos, memória cache L3 chegando a 40 MB,*

3) Memória principal:

O tamanho máximo da memória principal depende do registrador apontador de endereço de memória (IPR - Instruction Pointer Register).

=> arquitetura 32 bits, o IPR é de 32 bits, máximo de memória acessível é de 2^32 endereços

=> arquitetura 64 bits, o registrador IPR é de 64 bits, máximo de memória acessível é de 2^64 endereços.

4. (1,0) Descreva passo a passo as operações de leitura da memória e de escrita na memória, indicando como os registradores RDM e REM são utilizados e como a unidade de controle gera os sinais necessários.

Passos de uma operação de leitura

1) (REM) <- (outro registrador da UCP)
1.1) O endereço é colocado no barramento de endereços
2) Sinal de leitura é colocado no barramento de controle
2.1) Decodificação do endereço e localização da célula na memória
3) (RDM) <- (MP(REM)) pelo barramento de dados
4) (outro registrador da UCP) <- (RDM)

Passos de uma operação de escrita

1) (REM) <- (outro registrador)
1.1) O endereço é colocado no barramento de endereços
2) (RDM) <- (outro registrador)
2.1) O dado é colocado no barramento de dados
3) Sinal de escrita é colocado no barramento de controle

4) (MP(REM)) <- (RDM)

5. (1,0) Descreva como funciona uma arquitetura microprogramada e indique a diferença entre micro-instruções verticais e horizontais.

Em uma arquitetura microprogramada, a unidade de controle é especificada por um microprograma que consiste de uma seqüência de instruções de uma linguagem de microprogramação. Estas instruções são muito simples e especificam microoperações. Uma unidade de controle microprogramada é implementada com circuitos lógicos e é capaz de seguir uma seqüência de microinstruções gerando sinais de controle para que cada uma delas seja executada. Os sinais de controle gerados por uma microinstrução são usados para causar transferências de dados entre registradores e memória e execução de operações pela ULA.

As microinstruções horizontais tem como característica ter funções distintas para cada bit que a compõe, como por exemplo, controlar uma linha de controle interna da UCP, controlar uma linha de barramento externo de controle, definir condição de desviro e endereço de desvio entre outras. Tem a vantagem de ser simples e direto possível, podendo controlar várias microoperações em paralelo, além de uma eficiente utilização do hardware. E possui a desvantagem de maior ocupação de espaço de memória de controle em relação à microinstrução vertical.

As microintruções verticais se caracterizam por possuir um decodificador extra para identificar quais as linhas que serão efetivamente ativadas. Sua principal vantagem é reduzir o custo da Unidade de controle em função do menor tamanho da instrução, o que poderá ser necessário uma maior quantidade de instruções. Tem como principal desvantagem a redução do tempo devido à necessidade da decodificação dos campos de cada microinstrução.

6. (1,0) Considere a máquina apresentada na aula 4. Descreva detalhadamente (do mesmo modo que é apresentado na aula 4) como é realizada a execução das seguintes instruções:

a) LDA 230

a) RI <- Instrução lida
b) CI <- CI + 1
c) Decodificação do código de operação
- recebe os bits do código de operação
- produz sinais para a execução da operação de leitura em memória
d) Busca do operando na memória
- A UC emite sinais para que o valor do campo operando = 230 seja transferido para a REM
*- A UC emite sinais para que o valor contido no REM seja transferido para o barram*ento de endereços
- A UC ativa a linha READ do barramento de controle
- Conteúdo da posição da memória, conforme endereço contido no barramento de endereços (230), é transferido através do barramento de dados para o RDM
- O conteúdo do RDM é transferido para o registrador acumulador (ACC <- RDM)

b) JP 850

a) RI <- Instrução lida
b) CI <- CI + 1
c) Decodificação do código de operação
- recebe os bits do código de operação
- produz sinais para a execução da operação de salto condicional
d) UC emite sinal para transferir conteúdo acumulador para UAL (UAL <- ACC)
e) Executa operação de comparação
e.1) Resultado = verdadeiro, isto é, ACC > 0
CI <- Operando (CI <- 850)
d) Inicia o procedimento de leitura da instrução contida no endereço que consta em CI

7. (1,0) Escreva um programa que utilize as instruções de linguagem de montagem apresentadas na aula 4 para executar o seguinte procedimento. O conteúdo da memória cujo endereço é 80 é lido e verifica-se se o seu valor é maior que 0. Caso seu valor seja maior que 0, o conteúdo de memória cujo endereço é 100 é somado do conteúdo de memória cujo endereço é 80 e o resultado é armazenado no endereço 80. Caso contrário, o conteúdo de memória cujo endereço é 80 é multiplicado por 4 e o resultado é armazenado

no endereço 80. Além de apresentar seu programa escrito em linguagem de montagem, apresente também o programa traduzido para linguagem de máquina.

1ª. Opção de solução: nesta opção foram considerados os endereços de memória 80_{10} e 100_{10} na base 10 e na sequência de instruções foram utilizados os seus correspondentes em hexadecimal, 50_{16} e 64_{16}, respectivamente.

Endereço (hexa)	*Instrução*	*Descrição*	*Linguagem Máquina (bin / hexa)*
00	*LDA 50*	*ACC <- (50)*	*(0001 0101 0000 / 150)*
01	*JP 08*	*se ACC > 0, CI <- 08*	*(0110 0000 1000 / 608)*
02	*ADD 50*	*ACC <- ACC + (50)*	*(0011 0101 0000 / 350)*
03	*ADD 50*	*ACC <- ACC + (50)*	*(0011 0101 0000 / 350)*
04	*ADD 50*	*ACC <- ACC + (50)*	*(0011 0101 0000 / 350)*
05	*ADD 50*	*ACC <- ACC + (50)*	*(0011 0101 0000 / 350)*
06	*STR 50*	*(50) <- ACC*	*(0010 0101 0000 / 250)*
07	*HLT*	*Encerra Procedimento*	*(0000 0000 0000 / 000)*
08	*LDA 64*	*ACC <- (64)*	*(0001 0110 0100 / 164)*
09	*ADD 50*	*ACC <- ACC + (50)*	*(0011 0101 0000 / 350)*
0A	*STR 50*	*(50) <- ACC*	*(0010 0101 0000 / 250)*
0B	*HLT*	*Encerra Procedimento*	*(0000 0000 0000 / 000)*

2º. Opção de solução: Caso os endereços 80 e 100 sejam considerados em hexadecimal, o conjunto de instruções da aula 4 não poderá atender, em especial ao endereço 100_{16} . O último endereço que o operando de 8 bits adotado no conjunto de instruções da aula 4 pode ser utilizado é FF_{16}. A solução para este caso é aumentarmos o campo operando para 12 bits, consequentemente as instruções aumentarão para 16 bits, se for mantido o mesmo tamanho para o código de operação. As células da arquitetura apresentada na aula 4 tinham a capacidade de armazenamento de uma instrução e para manter essa mesma regra, de uma instrução em cada célula, cada célula deverá armazenar 16 bits.

Endereço (hexa)	*Instrução*	*Descrição*	*Linguagem Máquina (bin / hexa)*
000	*LDA 080*	*ACC <- (080)*	*(0001 0000 1000 0000 / 1080)*
001	*JP 008*	*se ACC > 0, CI <- 008*	*(0110 0000 0000 1000 / 5008)*
002	*ADD 080*	*ACC <- ACC + (080)*	*(0011 0000 1000 0000 / 3080)*
003	*ADD 080*	*ACC <- ACC + (080)*	*(0011 0000 1000 0000 / 3080)*
004	*ADD 080*	*ACC <- ACC + (080)*	*(0011 0000 1000 0000 / 3080)*
005	*ADD 080*	*ACC <- ACC + (080)*	*(0011 0000 1000 0000 / 3080)*
006	*STR 080*	*(080) <- ACC*	*(0010 0000 1000 0000 / 2080)*
007	*HLT*	*Encerra Procedimento*	*(0000 0000 0000 0000 / 0000)*
008	*LDA 100*	*ACC <- (100)*	*(0001 0001 0000 0000 / 1100)*
009	*ADD 080*	*ACC <- ACC + (080)*	*(0011 0000 1000 0000 / 3080)*
00A	*STR 080*	*(080) <- ACC*	*(0010 0000 1000 0000 / 2080)*
00B	*HLT*	*Encerra Procedimento*	*(0000 0000 0000 0000 / 0000)*

8. (2,0) Considere uma máquina com arquitetura semelhante àquela apresentada em aula. Pode-se endereçar no máximo 256 M células de memória, sendo que cada célula tem tamanho igual a 8 bits. Em cada acesso á memória, obtém-se o conteúdo de uma célula. Todas as instruções desta máquina possuem dois campos: o primeiro indica o código de operação e o segundo indica endereço de célula de memória onde se encontra o operando. Esta máquina possui 12 códigos de operação diferentes.

a) Calcule a capacidade mínima de endereçamento em bits do REM, considerando que os bits armazenados no REM são utilizados para endereçar uma célula de memória.

REM = Barramento de endereços, este terá a capacidade de endereçar 256M células = N

N = 256M células => $N = 2^{28}$ => e = 28 bits

REM = barramento de endereços = 28 bits

b) Calcule o número de bits que devem poder ser transmitidos no barramento de endereços em cada acesso à memória.

REM = barramento de endereços = 28 bits

c) Calcule o tamanho do RI (Registrador de Instruções).

RI terá que ter o tamanho de uma instrução

O tamanho da instrução = código de operação + operando.
Para atender a 12 códigos diferentes, o código de operação deverá ter no mínimo 4 bits
Tamanho da instrução = 4 + 28 = 32 bits => RI = 32 bits

d) Calcule o número de células que uma instrução ocupa.
Como cada instrução tem 32 bits, serão necessárias 4 células para armazenar 1 instrução.

e) Calcule a capacidade máxima de armazenamento da memória deste sistema em bits.
N = 256M células e M = 8 bits/célula
T = total de bits = N x M = 256M células x 8 bits/célula = 2^{28} x 2^{3} = 2^{31}bits

f) Calcule a capacidade mínima de endereçamento em bits do CI (Contador de Instrução), considerando que os bits armazenados no CI são utilizados para endereçar a primeira célula de uma instrução armazenada na memória.
Para ter a capacidade de endereçar todas as posições da MP dessa arquitetura, o CI deverá ter o tamanho mínimo de 28 bits, assim, poderá atingir a faixa de endereços de:
$00000000000000000000000000000_2$ (0_{10}) até
$11111111111111111111111111111_2$ ($268.435.455_{10}$)

9. (1,0) Considere uma máquina cujo relógio possui uma frequência de 6 GHZ e um programa P1 no qual são executadas 100 instruções desta máquina e um programa P2 no qual são executadas 2000 instruções.

6GHz = 6.000.000.000 Hz
Tempo de um ciclo de relógio = 1/6.000.000.000 = 0,000 000 000 167 seg ou 0,167ns (nanosegundos)

a) Calcule o tempo para executar os programas P1 e P2, considerando que cada instrução é executada em 6 ciclos de relógio e a execução de uma instrução só se inicia quando a execução da instrução anterior é finalizada.

Tempo de execução de 1 instrução = 6 ciclos de relógio x 0,167ns = 1,00 ns
Para o programa P1
*100 instruções executadas sequencialmente = 100 x 1,00ns = **100ns***
Para o programa P2
*2000 instruções executadas sequencialmente = 2000 x 1,00ns = **2000ns***

b) Uma nova implementação dessa máquina utiliza um pipeline de 5 estágios, todos de duração igual a 2 ciclos de relógio. Calcule o tempo para executar os programas P1 e P2, considerando que não existem conflitos de qualquer tipo.
Tempo para execução da 1ª instrução = 5 estágios x 2 ciclos x 0,167 ns por ciclo = 1,67ns
Para o programa P1
Tempo para execução das demais em pipeline = 99 x 2 x 0,167 = 33,07ns
Tempo para execução das 100 instruções =
1,67ns (primeira instrução) + 33,07ns (para as demais) = 34,74ns
Para o programa P2
Tempo para execução das demais em pipeline = 1999 x 2 x 0,167 = 667,7ns
Tempo para execução das 2000 instruções =
1,67ns (primeira instrução) + 667,7ns (para as demais) = 669,3ns

c) Calcule, para o programa P1 e para o programa P2, a relação entre os tempos de execução obtidos no item a com os tempos obtidos no item b.

Para o programa P1
100ns / 33,07ns = 3,02
O tempo gasto para executar as 100 instruções na arquitetura do item a é cerca de 3,02x maior que a da arquitetura do item b
Para o programa P2
2000ns / 669,3ns = 2,99.
O tempo gasto para executar as 2000 instruções na arquitetura do item a é cerca de 2,99x maior que a da arquitetura do item b

Concluindo, a arquitetura contendo pipeline (item b) tem desempenho melhor que uma arquitetura sem esse recurso(item a) e o desempenho melhora conforme aumenta a quantidade de instruções que compõe o programa.