

P28) A velocidade do enlace é menor que a do envio, o que vai gerar um acúmulo no buffer de entrada de A, podendo ocorrer perdas de dados.

→ Arquitetura 2 - Prova Georica - Emanuel Henrique - 473360

② O núcleo ARM não é uma arquitetura RISC pura por conta das restrições de sua aplicação principal, o sistema embarcado. Em algum sentido, a força do núcleo ARM é que ele não leva o conceito RISC longe demais. Nos sistemas de hoje, a chave não é a velocidade bruta do processador, mas o desempenho efetivo e consumo de energia do sistema. O conjunto de instruções ARM difere da definição RISC pura de várias formas que tornam o conjunto inadequado para embarcado: Ciclo de execução variável para certas instruções, nem todas as instruções ARM são executadas em um único ciclo. Por exemplo, as instruções load-store-multiple variam no número de ciclos dependendo do número de registradores envolvidos. Barrel shifter embutido levando instruções complexas, normalmente usado para deslocar e girar n-bits dentro de um único ciclo de clock. Conjunto de instruções Thumb de 16 bits que permite ao núcleo ARM executar instruções de 16 ou 32 bits. Execução condicional, recurso que melhora o desempenho e a densidade do código, reduzindo branches. Essas são algumas justificativas da ARM não ser puramente RISC.



④ Uma pipeline é o mecanismo que um processador RISC usa para executar instruções. O seu uso acelera a execução buscando a próxima instrução enquanto outra está sendo decodificada e executada. O pipeline do ARM não processa uma instrução até que ela passe completamente pelo estágio de execução, ou seja, executa uma instrução somente quando a quarta instrução for buscada. Mesmo no pipeline, o PC está sempre apontando 8 bytes adiante a instrução atualmente em execução.



⑤ O preíndice com writeback calcula um endereço de um registrador base, mais o offset de endereço e em seguida, atualiza esse registrador base com o novo endereço. Em contraste, o préíndice sem writeback é igual ao préíndice com writeback, mas não atualiza o registrador base com o novo endereço. O postíndice atualiza o registrador base somente após o endereço ser usado.

⑥ SPSR é o registrador de status do programa, recebe como valor atual do CPSR quando uma exceção é feita, de modo que o CPSR possa ser restaurado após o tratamento da exceção ou se for exceção. Não está disponível nos modos USER e SYSTEM, pois ambos os modos não tratam exceções.

①  $R10 = 0x55aabb44$ ;  $R11 = 0x00001234$ ;  $R12 = 0x00001000$   
 $R13 = 0x2000e000$

a) STMED  $sp!$ ,  $r10-12$

Antes da execução

Endereço	Valor
$0x40018$	$0x00000000$
$sp \rightarrow 0x40014$	$0x00000001$
$0x40010$	-
$0x4000c$	-
$0x40008$	-

Depois da execução

Endereço	Valor
$0x40018$	$0x00000000$
$0x40014$	$0x00000001$
$0x40010$	$0x00001000$
$0x4000c$	$0x00001234$
$sp \rightarrow 0x40008$	$0x55aabb44$

b) STMFA  $sp!$ ,  $r10$

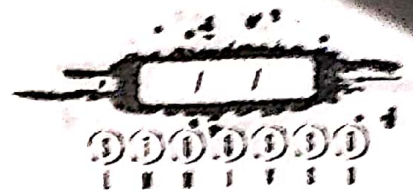
Antes

Endereço	Valor
$0x40018$	$0x00000001$
$P \rightarrow 0x40014$	$0x00000000$
$0x40010$	-

Depois

Endereço	Valor
$0x40018$	$0x00000000$
$0x40014$	$0x00000001$
$0x40010$	$0x55aabb44$
$sp \rightarrow 0x2000c$	-





c) STMIB r10!, r11, r12

Antes

$R10 = 0x55aa\,bb44$

$R11 = 0x0000\,1234$

$R12 = 0x0000\,1000$

Depois que armazena R12:

$R10 = 0x55aa\,bb48$

Depois que armazena R11:

$R10 = 0x55aa\,bb4c$

\* A instrução STMIB usa armazenamento na memória os valores de R11 e R12 incrementando o endereço de R10 antes (IB = increment before).

d) STMIA r10!, r11, r12

◊ STMIA armazena na memória os conteúdos de R11 e R12 apontados pelo "r10". A atualização de r10 ocorre após o armazenamento do conteúdo nos registradores na memória.

Antes

Depois

$SP = 0x40010 \Rightarrow$  Armazena R12 na memória  $\Rightarrow r10 = 0x40014$

$\Rightarrow$  Armazena R11 na memória  $\Rightarrow r10 = 0x40018$

e) STR r10, [r12]

A instrução armazena o conteúdo de r10 no endereço apontado por r12.

Antes

Depois

$\text{mem32}[0x0000\,1000] = ???$

$\text{mem32}[0x0000\,1000] = \text{conteúdo r10}$

$\text{mem32}[0x0000\,1000] = 0x55aa\,bb44$

f) STR r11, [r10, r12]

Antes de executar

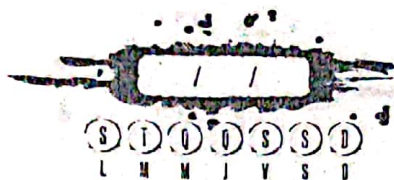
Nessa instrução, R12 atua como offset:  $\text{mem32}[0x55aa\,bb44 + 0x0000\,1000] = ???$

Depois:  $\text{mem32}[0x55aa\,bb44] = 0x0000\,1234$

↳ R12 offset

e  $R10 = 0x55aa\,bb44$ . O "!" significa que o endereço + offset calculado para escrita no registrador base, R10.





g) STR R10, [R12], #4

Antes: R12 = 0x00001000

Depois: mem32[0x00001000] = 0x55aabb44

mem32[0x00001000] = ???

R12 = 0x00001004

h) STMED r0!, r10-r12

Antes		Depois	
0x40018	0x00000000	0x40018	0x00000000
0x40014	-	0x40014	0x00001000
		0x40010	0x00001234
		0x4000c	0x55aabb44
		0x40008	-

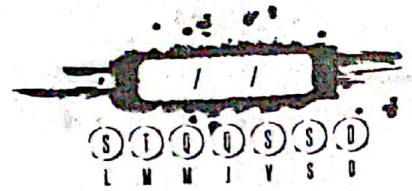
ED → Empty descending stack

mp →

③ O processador muda para um modo de solicitação de interrupção específica, que reflete a interrupção sendo gerada. O valor contido em CPSR é usado em um SPSR do novo modo de solicitação de interrupção. O PC é usado em um LR do novo modo de solicitação de interrupção. Interrupções são desabilitadas - IRQ e FIQ são desabilitados no CPSR, impedindo outra solicitação do mesmo tipo. O processador define o PC para um endereço de memória que está dentro de um intervalo especial que representa a tabela de vetores. Após isso, o processador retorna para a tabela, suspende a execução normal, trata a interrupção e restaura o contexto anteriormente salvo em SPSR e retorna para a execução normal do programa.

⑦ É um dispositivo de hardware formado a partir de uma pequena memória que fica junto da MMU do processador. Sua função é facilitar a tradução de endereços virtuais em físicos, com o objetivo de evitar consulta à tabela de páginas presente na memória. Um endereço virtual é enviado para a MMU, ela faz uma comparação do endereço recebido com as entradas de tabela presentes na TLB. Caso esse endereço esteja na TLB, paga imediatamente de lá, se não, vai na memória. Por assim como o cache, a TLB acelera o acesso à memória.





8a) Cada conjunto possui 1024 linhas, como são 4 conjuntos, então:  
 $1024 \cdot 4 = 4096$  linhas. Cada linha possui 128 bits, então:  $4096 \cdot 128 = 524288$   
bits no total.  $\frac{524288}{8} = 65536$  Bytes = 64KB

b) O responsável pela ~~endereços~~ endereços de byte é o ~~data index~~ data index. Cada linha tem 16B de tamanho sendo necessários 4 bits para ~~endereços~~ endereços e endereçamento,  $2^4 = 16$ , os bits usados serão 0, 1, 2 e 3.

c) ~~Cada linha possui 1024~~ Cada conjunto possui 1024 linhas, como  $2^{10} = 1024$ , então são necessários 10 bits para endereçá-las. Os bits 4-13 são responsáveis por ~~isso~~ e são chamados de ~~net index~~ net index.

d) Os bits responsáveis pela ~~isso~~ os 18 bits restantes, bits 14-31.

e) Para ~~isso~~, é preciso olhar para os bits 4-13 que fazem parte do ~~net index~~ net index, para que fazem parte da mesma linha os bits correspondentes no endereço devem ser iguais:  $0x79CC3FF0$ ,  $0xAAC0FFF3$ ,  $0xFCB17FF7$ ;  $0xA13DBFEA$

10. Dão 4 pré-condições e 4 instruções da esquerda para a direita:

Instrução 1:  $LDMIA\ R0!, \{R1-R3\}$ ; Instrução 2:  $LDMIB\ R0!, \{R1-R3\}$ ; Instrução 3:  $STMDB\ R0!, \{R1-R3\}$ ; Instrução 4:  $STMIA\ R0!, \{R1-R3\}$

9) A cache demonstrada pela questão é uma cache ~~associativa~~ associativa dividida em quatro conjuntos iguais. O campo ~~net index~~ net index serve para direcionar a linha em cada conjunto. Então, para verificar se o dado pedido pela CPU está presente na cache é feita uma comparação: é comparado o tag de cada linha na ~~cache~~ cache com o tag do endereço representado pela CPU. Se os tags ~~botarem~~ botarem então o dado está presente na cache e a transferência ~~para~~ da memória para a CPU é realizada, caso não esteja presente, o endereço é buscado na memória principal e ~~em~~ guardado na cache.