(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 Teórica – Samuel Henrique – 473360

② O núcleo ARM não é uma arquitetura RISC pura por conta das restrições de sua aplicação principal, os sistema embarcado. Em algum sentido, a força do núcleo ~~do~~ 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 adequado para embarcados: 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 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 o núcleo ARM executar instruções de 16 ou 32 bits. Execução condicional, recurso que melhora o desempenho e da densidade do código, reduzindo branchs. Essas não algumas justificativas da ARM não ser puramente RISC.

④ Um pipeline é o mecanismo que um processador RISC usa para executar instruções. O meu usa cerca de execução buscando a próxima instrução enquanto outras estão sendo decodificadas e executadas. 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. Nesse do pipeline, o PC está sempre apontando 8 bytes adiante a instrução atualmente em execução.

/ /

S T Q Q S S D
L M M I V S D

⑤ O preindex 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 ~~funcionamento~~ do preindex é igual ao preindex com writeback, mas não atualiza o registrador base com o novo endereço. O postindex atualiza o registrador base somente após o endereço ser usado.

⑥ SPSR é registrador de status do programa salvo armazena o valor atual do CPSR quando uma exceção é feita, de modo que o CPSR possa ser restaurado após o tratamento da exceção ou ~~de~~ das exceções. 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) STMFD sp!, r10-12

Antes da execução

	Endereço	Valor
	0x40018	0x00000000
sp →	0x40014	0x00000001
	0x40010	-
	0x4000c	-
	0x40008	-

Depois da execução

	Endereço	Valor
	0x40018	0x00000000
	0x40014	0x00000001
	0x40010	0x00001000
	0x4000c	0x00001234
sp →	0x40008	0x55aabb44

b) STMEA sp!, r10

Antes

	Endereço	Valor
	0x40018	0x00000001
p →	0x40014	0x00000000
	0x40010	-

Depois

	Endereço	Valor
	0x80018	0x00000000
	0x80014	0x00000001
	0x80010	0x55aabb44
sp →	0x2000c	-

e) STMIB r10!, r11, r12

Antes

R10 = 0x55aabb44

R11 = 0x00001234

R12 = 0x00001000

Depois que armazena R12:

R10 = 0x55aabb48

Depois que armazena R11:

R10 = 0x55aabb4C

* A instrução STMIB vai armazenar na memória os valores de R11 e R12 incrementando o endereço de R10 antes (IB = increment before)

d) STMIA sp!, r11, r12

O STMIA armazena na memória os conteúdos de R11 e R12 apontado pelo "sp". A atualização do sp ocorre após o armazenamento do conteúdo dos registradores na memória.

Antes

SP = 0x40010 ⇒ Armazena R12 na memória ⇒ sp = 0x40014

⇒ Armazena R11 na memória ⇒ sp = 0x40018

c) STR r10, [r12]

A instrução armazena o conteúdo de r10 no endereço apontado por R12.

Antes

mem32[0x00001000] = ???

Depois

mem32[0x00001000] = conteúdo r10

mem32[0x00001000] = 0x55aabb44

f) STR r11, [r10, r12]!

Antes de executar

Nessa instrução, R12 atua como offset: mem32[0x55aabb44 + 0x00001000] = ???

↳ R12 offset

Depois: mem32[0x55aabb44] = 0x00001234

e ~~R10 = 0x55aabb~~ R10 = 0x55aabb44. A "!" significa que o endereço + offset calculado será escrito no registrador base, R10.

g) STR R10, [R12], #4

Antes: R12 = 0x00001000
mem32[0x00001000] = ???

Depois: mem32[0x00001000] = 0x55aabb44
R12 = 0x00001004

h) STMED sp!, R10-R12

Antes

	0x40018	0x00000000
sp →	0x40014	—

Depois

	0x40018	0x00000000
	0x40014	0x00001000
	0x40010	0x00001934
	0x4000C	0x55aabb44
sp →	0x40008	—

ED → Empty descending stack

③ O processador muda para um modo de solicitação de interrupção específico, que reflete a interrupção vinda gerada. O valor contido em CPSR é salvo no SPSR do novo modo de solicitação de interrupção. O PC é salvo no LR do novo modo de solicitação de interrupção. Interrupções são desabilitadas - IRQ e FIQ são desabilitadas 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. Caso ocorra o processador pula 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 da tabela presentes na TLB. Caso esse endereço esteja na TLB, pega diretamente de lá, se não, vai na memória. Tal qual como as caches, a TLB salva o acesso à memória

/ /

SEGUNDA

LMIYSD

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

b) O ~~resposável~~ responsável pela ~~endereço~~ endereço do byte é o Byte index. Cada linha tem 16B de tamanho sendo necessários 4 bits para ~~endereçá-los~~ ~~endereçar~~ o endereçamento, $2^4 = 16$, os bits usados serão 0, 1, 2 e 3.

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

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

e) Para isso, é preciso olhar para os bits 4-13 que fazem parte do set index, para que façam parte da mesma linha os bits correspondentes no endereço devem ser iguais: 0x79CC3FF0, 0xAACOFFF3, 0xFCB17FF7, 0xA13DBFFA

⑩ São 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 dividida em quatro conjuntos iguais. O campo set 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 a tag de cada linha na cache com a tag do endereço apresentado pela CPU. Se ~~os~~ as tags baterem então o dado está presente na cache e a transferência ~~para~~ da palavra para a CPU é realizada, caso não esteja presente, o endereço é buscado na memória principal e ~~guardado~~ guardado na cache.