Nome: Anderson Silva Souza
Matrícula: 475242

Prova teórica
Arquitetura e organização de computadores II

(1) R10 = 0x55aabb44
R11 = 0x00001234
R12 = 0x00001000
R13 = 0x20008000

(a) STMFD sp!, r10-r12

ANTES DA EXECUÇÃO

Endereço	Valor
0x 20018	0x00000000
0x 20014	0x00000001
0x 20010	–
0x 2000C	–
0x 20008	–

sp →

DEPOIS DA EXECUÇÃO

Endereço	Valor
0x 20018	0x00000000
0x 20014	0x00000001
0x 20010	0x00001000
0x 2000C	0x00001234
0x 20008	0x 55aabb44

sp →

(b) STMEA sp!, r10

ANTES DA EXECUÇÃO

Endereço	Valor
0x 20018	0x00000000
0x 20014	0x00000001
0x 20010	–

sp →

DEPOIS DA EXECUÇÃO

Endereço	Valor
0x 20018	0x00000000
0x 20014	0x00000001
0x 20010	0x 55AABB44
0x 2000C	–

sp →

(c) STMIB r10!, r11, r12

ANTES DA EXECUÇÃO

$R_{10} = 0x55AABB44$

$R_{11} = 0x00001234$

$R_{12} = 0x00001000$

A instrução STMIB vai armazenar na memória os valores de R_{11} e R_{12}.

DEPOIS QUE ARMAZENA O VALOR DE R_{12} NA MEMÓRIA

- $R_{10} = 0x55AABB48$

DEPOIS QUE ARMAZENA O VALOR DE R_{11} NA MEMÓRIA

- $R_{10} = 0x55AABB4C$

(d) STMIA sp!, r11, r12

A instrução STMIA vai armazenar na memória os conteúdos de R_{11} e R_{12} apontado pelo "sp". A atualização de sp ocorre após o armazenamento na memória.

ANTES DA EXECUÇÃO

- SP = 0x20010

DEPOIS QUE ARMAZENAR O CONTEÚDO DE R_{12} NA MEMÓRIA

- SP = 0x20014

DEPOIS QUE ARMAZENAR O CONTEÚDO DE R_{11} NA MEMÓRIA

- SP = 0x20018

② STR R10, [R12]

A instrução STR vai armazenar o conteúdo de R_{10} no endereço de memória apontado pelo R_{12}

ANTES DA EXECUÇÃO

mem32[0X00001000] = ?

DEPOIS DA EXECUÇÃO

mem32[0X00001000] = conteúdo de R10

mem32[0X00001000] = 0X55AABB44

① STR R11, [R10, R12]!

ANTES DA EXECUÇÃO

mem32[0X55AABB44 + 0X00001000] = ?

0X55AABB44 → R10

0X00001000 → R12 OFFSET

DEPOIS DA EXECUÇÃO

mem32[0X55AACB44] = 0X00001234 → R11

R10 = 0X55AACB44

O "!" significa que o endereço calculado será escrito no registrador de endereço base, ou seja, no R10.

(2) Temos algumas justificativas para o ARM não ser considerado puramente RISC. Uma preferência que ~~são~~ estão sendo desenvolvidas, tem a necessidade do processador ser mais rápido no processamento das informações, mas principalmente é preciso que esses processadores tenham um baixo consumo de energia. Em sua ~~seu~~ principal utilização, sistemas embarcados, ele deixa de ser puramente RISC porque algumas instruções levam mais de um ciclo para ser executadas. Como é o caso das instruções que carregam ou armazena em múltiplos registradores.

(3) Uma interrupção é um evento que desvia o fluxo de execução de instruções para uma posição prefixada de memória onde será tratada uma rotina de interrupção. Quando ocorre uma interrupção, o CPU determina o PC para um endereço de memória. Esse endereço está contido na tabela de vetores e podemos ter acesso a este endereço, pelo offset. Quando o processador vai tratar uma interrupção, ele suspende a execução do programa e dar início a rotina de interrupção selecionada na tabela de vetores.

(4) Pipeline, semelhante a uma linha de produção, é uma forma que os processadores ARM utiliza para executar as instruções. Esse mecanismo tem como finalidade executar a próxima instrução enquanto outra está sendo decodificada, sendo assim, acelerando o processo de execução do conjunto de instruções. ~~O pipeline do arm tem uma característica que não permite a execução de o processamento.~~

(5)

PREINDEX WITH WRITEBACK

- Realiza um cálculo do registrador com endereço base + offset e coloca ~~em~~ no registrador que contém o endereço base o resultado da soma. Portanto, o endereço base será atualizado.

PREINDEX

- Realiza o cálculo do registrador com endereço base + offset, mas o registrador que contém o endereço base não será atualizado com o valor da soma.

~~POST~~
POSTINDEX
- Apenas atualiza o registrador que contém o endereço base depois de sua utilização.

(6) O SPSP é um registrador de status. Quando temos uma exceção, o valor de CPSP é salvo nele (SPSR). Ele não está disponível ~~nos~~ porque nos modos USER e ~~SYT~~ SYSTEM não fazem o tratamento de exceções.

(7) Translation Lookaside Buffer (TLB) é um dispositivo de hardware ~~utilizado~~ com a finalidade de realizar o mapeamento de endereços virtuais em endereços físicos sem a ~~ne~~ necessidade de acessar a tabela de página. A TLB fica dentro da MMU. Seu funcionamento é da ~~seguinte~~ seguinte forma, quando um programa envia um endereço virtual à MMU, o hardware compara o endereço recebido com todas as entradas da tabela TLB de forma simultânea. Caso encontre, esta página é carregada diretamente da TLB. Sua principal função é acelerar o acesso à memória.

(9) Podemos observar que se trata de uma cache associativa em conjunto. Essa forma de organizar a cache é importante para reduzir a frequência de ~~thr~~ THRASHING. Dividindo a cache em quatro conjuntos iguais. Os bits 4, 5, 6, 7, 8 e 9 do endereço, chamados de "set index" servem para selecionar uma linha ~~dos quatro conjunto~~ em cada um dos quatro conjuntos. Quando uma informação ~~for armazenada~~ precisa ser armazenada ela pode ser escrita em qualquer linha selecionada pelo set index do endereço. Basicamente esse é o funcionamento de uma cache associativa em conjunto.

(10)

(1)

Pré - condição

$r_0 = 0x00080010$

$r_1 = 0x00000000$

$r_2 = 0x00000000$

$r_3 = 0x00000000$

Instrução

LDMIA R0!, {R1 - R3}

Pós - condição

$r_0 = 0x0008001C$

$r_1 = 0x00000001$

$r_2 = 0x00000002$

$r_3 = 0x00000003$

(2)

Pré - condição

$r_0 = 0x00080010$

$r_1 = 0x00000000$

$r_2 = 0x00000000$

$r_3 = 0x00000000$

Instrução

LDMIB R0! {R1 - R3}

Pós - condição

$r_0 = 0x0008001C$

$r_1 = 0x00000002$

$r_2 = 0x00000003$

$r_3 = 0x00000004$

(3)

Pré - condição

$r_0 = 0x0008001C$

$r_1 = 0x00000002$

$r_2 = 0x00000003$

$r_3 = 0x00000004$

Instrução

STMDB R0!, {R1 - R3}

Pós - condição

$r_0 = 0x00080010$

mem32[0x80010] = 0x02

mem32[0x80014] = 0x03

mem32[0x80018] = 0x04

(4)

Pré - condição

$r_0 = 0x00080014$

$r_1 = 0x00000001$

$r_2 = 0x00000002$

$r_3 = 0x00000003$

Instrução

STMIA R0!, {R1 - R3}

Pós - condição

$r_0 = 0x00080020$

mem32[0x8001C] = 0x03

mem32[0x80018] = 0x02

mem32[0x80014] = 0x01

8

(a) Como cada conjunto tem 1024 linhas, ao todo teremos $1024 \cdot 4 = 4096$ linhas. Temos que cada linha tem 128 bits, $4096 \cdot 128 = 524288$ bits. Dividindo por 8, temos a quantidade em bytes.

$\frac{524288}{8} = 65536$ Bytes ou 64KB.

(b) Cada linha tem um tamanho de 16 BYTES e são necessários 4 bits, ou seja, $2^4 = 16$. Logo, os bits 0, 1, 2 e 3 são do data index, que são responsáveis pelo endereçamento do byte.

(c) Temos que cada conjunto tem 1024 linhas. Para endereçá-los é necessário 10 bits, ou seja, $2^{10} = 1024$. Os bits responsáveis por determinar uma linha do cache são: 4, 5, 6, 7, 8, 9, 10, 11, 12 e 13 chamados de set index.

(d) Os bits responsáveis pela Tag são: 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30 e 31. 18 bits para a Tag.

(e) Quem determina a linha do cache é o campo set index. Portanto, seus bits devem serem iguais. Logo,

0X73ABFAA7 0X14A3FAA7

0X9CE31AA7 0X32CB7AA7