|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | PES University, Bangalore  (EstablishedunderKarnatakaActNo.16of2013)  **UE22CS251B: Microprocessor and Computer Architecture Question Bank: Subjective Type** | | | |
| **Sl.No** | | **Topic** | | | Question | **Grade** | |
|  | | **ARM 7TDMI:**  **Unit-1** | | | 1. Given that LDR R1 ,=0x10000000 ;starting address of data transfer, R4 contains 0x10,R5 contains 0x20,R6 contains 0x30   What will be the content of register R1 and memory locations 0X10000004,0X10000008 and 0X1000000C after the execution of the following instruction **STMIA R1! , {R4-R6 }**   1. Given that LDR R1 ,=0x10000000 ;starting address of data transfer, R4 contains 0x10,R5 contains 0x20,R6 contains 0x30   What will be the content of register R1 and memory locations 0X10000000,0X10000004 and 0X10000008 after the execution of the following instruction **STMIB R1! , {R4-R6 }** | **Medium** | |
|  | | **ARM 7TDMI:**  **Unit-1** | | | Write a program in ARM7TDMI-ISA to achieve the following.  Copy blocks of 1024 bytes from source address A to a destination address B. Given that register **R9** points to the start of the source data while register **R10** points to the start of the destination location. Register **R11** indicates the end of the source data.  { Hint: Use 8 General Purpose Registers to transfer once.} | **Medium** | |
|  | | **ARM 7TDMI:**  **Unit-1** | | | Explain how stack concept is used for passing parameters in ARM. Write an ARM assembly language instruction(s) to push R4 onto the program stack referenced by SP and pop the element from SP to be copied in R5.  **Solution:** The Memory access can also be done in LIFO order i.e Can be treated like STACK. We have a portion of memory behaving like stack. R13 is a stack pointer which will keep the address of TOP of the STACK. Mainly used in Procedural Call. Stack can grow upward or downward direction based on the MODE used by the user in the program.  The Multiple data transfer instructions provide a mechanism for storing state on the runtime stack (pointed to by the stack pointer, r13 or sp). STM instruction is used to PUSH on to the STACK whereas  LDM is used to POP from the STACK.  Instruction to push the value in stack STMFD R13! {R4} or STMFD SP! {R4}  Instruction to Pop the value from stack LDMFD R13! {R5} or LDMFD SP! {R5} | **Medium** | |
|  | | **ARM 7TDMI:**  **Unit-1** | | | Given the following code snippet in C programming language.  **i=1;**  **While( i<=8)**  **{**  **g = h + A[i] ;**  **}**  g: Register R1, h: Register R2, Register R3: base address of A  use LDR and ADD instructions to convert this to ARM Assembly language | **Easy** | |
|  | | **ARM 7TDMI:**  **Unit-1** | | | What are banked registers? Show how the banked registers are utilized when the user mode changes to IRQ mode. | **Medium** | |
|  | | **ARM 7TDMI:**  **Unit-1** | | | Consider the register R0 and R1 as input registers which branches to subroutine swap. Write a program for above scenario which swap 2 numbers without using SWP instruction. Let the starting address of program of 32 bit architecture be 0X00003000 . Explain how the contents of register LR and PC change when Branch instruction is executed in the program and control is returned back to the program.  **ANS:**  .text  0X00003000 Mov R0,#10  0X00003004 Mov R1, #20  0X00003008 STMFD R13!,{r0,r1}  0X0000300C BL SWAP  0X00003010 SWI 0X011  0X00003014 SWAP:  0X00003014 LDMFD R13!,{r0,R1}  0X00003018 Add r0,r0,r1  0X0000301C Sub r1,r0,r1  0X000020 Sub r0,r0,r1  0X00003024 BX LR  The address of label and LDMFD is same.  When BL swap is executed:   1. PC=0X00003010 2. LR=0X00003010   PC=0X00003014  When control is returned back BX LR  PC=0X00003010 |  | |
|  | | **ARM 7TDMI:**  **Unit-1** | | | What is the hexadecimal representation of the instruction “BEQ Assignment”. Indicate the address of all instructions in program. Assume the starting address of program as 0X0000103C. Given  Instruction format and condition code for always=1110 and EQ=0000    .text  mov r1, #3  BEQ assignment  add r2, r0, r1  add r2,r0,r1  add r0,r0,#1 swi 0x011  Assignment:  mov r0, #2  bx lr  .end  Answer:  0X0000103C mov r1, #3  0X00001040 BEQ assignment  0X00001044 add r2, r0, r1  0X00001048 add r2,r0,r1  0X0000104C add r0,r0,#1  0X00001050 swi 0x011  0X00001054 Assignment:  0X00001054 mov r0, #2  0X0000105C bx lr  .end  Cond:0000  L=0  Offset =3  000010100000…..011=0X0A000003 | **Medium** | |
|  | | **ARM 7TDMI:**  **Unit-1** | | | Describe the process used for passing parameter in the ARM procedure call. Using an ALP procedure call standard, write a subroutine MAX that takes two parameters a and b and returns the larger of the two.  Solution: Block transfer instructions can be used to pass parameters from set of registers or stack memory to the procedure called.  .data  A: .Word 10  B: .Word 20  C: .Word 0  .text  LDR R0, =A  LDR R1, =B  LDR R2, =C  LDR R3, [R0]  LDR R4, [R1]  STMFD R13!, {R3, R4}  BL MAX  STR R5, [R2]  SWI 0X11  MAX:  LDMFD R13!, {R6, R7}  CMP R6, R7  BGT LOOP2  MOV R5, R7  MOV PC, LR  LOOP2:  MOV R5, R6  MOV PC, LR | **Difficult** | |
|  | | **ARM 7TDMI:**  **Unit-1** | | | Explain how stack concept is used for passing parameters in ARM. Write an ARM assembly language program demonstrating the use of stack in nested procedure calls. Also, use instructions LDM and STM for push and pop operations.  Solution: The Memory access can also be done in LIFO order i.e Can be treated like STACK. We have a portion of memory behaving like stack. R13 is a stack pointer which will keep the address of TOP of the STACK. Mainly used in Procedural Call. Stack can grow upward or downward direction based on the MODE used by the user in the program.  The Multiple data transfer instructions provide a mechanism for storing state on the runtime stack (pointed to by the stack pointer, r13 or sp). STM instruction is used to PUSH on to the STACK whereas  LDM is used to POP from the STACK.  .TEXT        **; MAIN Procedure**  LDR R4, =A  MOV R1, #11  MOV R2, #10  MOV R3, #02  STMFD R13!, {R1,R2,R3}  **BL  SUM        ; Call to ADD Procedure**  STR R0, [R4]  SWI 0x11  .DATA  A:    .WORD  0  **SUM:**  LDMFD R13!, { R4, R5,R6}       **; ADD Procedure**                    ADD R0, R4, R5                    STMFD R13!, {R0,R6,LR}  **BL MUL                                ; Call to MUL Procedure**    **MUL:**  LDMFD R13!, { R4, R5,LR}                     MUL R0, R4, R5                     MOV PC, LR                           **; Return to main procedure** | **Medium** | |
|  | |  | | | Consider the following sequence of instructions in MIPS architecture.  LDR R1, [R6,#40]  BEQ R2, R3, LABEL2 ; BRANCH TAKEN  ADD R1, R6, R4  LABEL2:BEQ R1,R2, LABEL1 ; BRANCH NOT TAKEN  STR R2,[R4, #20]  AND R1, R1, R4  a. Draw the pipeline execution diagram for this code, assuming there are  no delay slots and that branches execute in the EX stage.  b. Repeat the exercise mentioned in a and draw the pipeline execution  diagram for this code, assuming that delay slots are used by writing a  “SAFE INSTRUCTION” in the delay slot. |  | |
| **Unit-2 Pipelining** | | | | | | | |
| **1.** | **Pipelining:**  **Structural Hazard : Unit-2** | | Which combinations of the instructions below are involved in a Structural Hazard when handled by a 5-stage pipeline processor?   1. Instruction 1: add R2, R3, R4 2. Instruction 2:  ldr R5,[R6] 3. Instruction 3: sub R7, R8, R9 4. Instruction 4:  sub R10,R11, R1 5. Instruction 5: str R2, [R12] | | | | **Medium** |
| **2.** | **Pipelining: Structural Hazard :**  **Unit-2** | | Observe the following figure. In any system, instruction is fetched from memory in IF machine cycle. In the 4-stage pipeline shown Result Writing (RW) may access memory or one of the General-Purpose Registers depending on the instruction. At t4, Instruction-1(I1) is at RW stage and Instruction-4(I4) at IF stage. Alas! Both I1 and I4 are accessing the same resource i.e memory if I1 is a STORE instruction. Explain if it is possible to access memory by 2 instructions from the same CPU in a timing state? What dependency does it indicate? What is the solution?  Structural dependency scenario | | | | **Medium** |
| **3.** | **Pipelining: Structural Hazard :**  **Unit-2** | | Consider an instruction pipeline for the MIPS32 processor where data references constitute 42% of the instructions, and the ideal CPI ignoring memory structural hazards is 1.25. How much faster is the ideal machine without the memory structural hazard versus the machine with the hazard? | | | | **Medium** |
| **4.** | **Pipelining:**  **Unit-2** | | Loads/stores (MEM) use same memory port as instruction fetches (IF).Given that 30% of all instructions are loads and stores. Assume CPIold is 1.5  1 2 3 4 5 6 7 8 9  i IF ID EX MEM WB <— a load  i+1 IF ID EX MEM WB  i+2 IF ID EX MEM WB  i+3 \*\* IF ID EX MEM WB  i+4 IF ID EX MEM WB  How much faster could a new machine with two memory ports be? | | | |  |
| **5.** | **Pipelining: Data Hazard :**  **Unit-2**  **Pipelining: Data Hazard :**  **Unit-2** | | Examine how data dependencies affect execution in the basic 5-stage pipeline processor. The execution of the branch instruction happens in the execute stage.  **LW $1, 40($6)**  **ADD $6, $2, $2**  **SW $6, 50($1)**  i. Assuming there is no forwarding in this pipelined processor,  Indicate the hazards and add NOP instructions to eliminate the same.  ii. Assuming there is full forwarding,  Indicate the hazards and add NOP instructions to eliminate the same.  Show the same using pipeline execution diagrams.  **Answer:**  **Case i.**  LABEL1: LW $1, 40 ($6) IF ID EX MEM WB  BEQ $2, $3, LABEL2 ; BT IF ID EX MEM WB  **ADD $1, $6, $4** IF ID FLUSHES………..  ( dependency is seen)  LABEL2: BEQ $1, $2, LABEL1 ; BNT IF ID EX MEM WB  SW $2, 20 ($4) IF ID EX MEM WB  AND $1, $1, $4 IF ID EX MEM WB  **Case ii.**  LABEL1: LW $1, 40 ($6) IF ID EX MEM WB  BEQ $2, $3, LABEL2 ; BT IF ID EX MEM WB  **AND $1, $1, $4** IF ID EX MEM WB  ADD $1, $6, $4 IF ID EX MEM WB  ( safe instruction is written after branch taken avoids flushing)  LABEL2: BEQ $1, $2, LABEL1 ; BNT IF ID EX MEM WB  SW $2, 20 ($4) IF ID EX MEM WB | | | | **Medium** |
| **7.** | **Pipelining Branch Prediction:**  **Unit-2** | | Consider a program with the following behavior.  T, T, T, NT, NT, T, T, T, T, NT, NT, T ( where NT –branch Not Taken & T – branch Taken).  How many miss predictions are seen if the initial state is strongly taken for a 2 bit prediction?  **Answer:**  Given behavior :  T, T, T, NT, NT, T, T, T, T, NT, NT, T  Initial State: Strongly Taken (T).  Outcome:  T T T T T NT NT T T T T NT  **No of miss predictions are: 7** | | | |  |
| **8.** | **Pipelining Hazards :**  **Unit-2** | | What are the different types of hazards observed during pipelining? Explain how to overcome the same.  **Answer:**   * ***Structural Hazard:***    + Due to structure of the pipeline   + Hardware cannot support combination of instructions in the pipeline needing the same resources   Solution: Additional Hardware is required,  Harvard Architecture for memory.   * ***Data Hazard:***    + Due to data dependencies between instructions   + Instruction depends on the result of prior instruction still in the pipeline   In Software  Solution 1: Re-order instructions  Solution 2: insert independent instructions (or no-ops) Ex: MOV R0, R0  In Hardware  Solution 1: Insert bubbles (i.e. stall the pipeline)  Solution 2: Data Forwarding   * ***Control Hazard:***    + Due to control instructions   + Pipelining of branches & other instructions that change the PC   Safe Instruction, Delayed Branching, Bringing the execution of branch instruction earlier in the pipelining, Branch Prediction. | | | | **Medium** |
| **9.** | **Pipelining Hazards :**  **Performance:**  **Unit-2** | | Given that two programs P1 and P2 have an ideal CPI value equal to 1.For P1 , 10% of the instructions have a load-use hazard and 15% of its instructions aretaken branches. For P2 , 20% of the instructions have a load-use hazard, and 5% of its instructionsare taken branches.  (Note: For a load use hazard one stall cycle is needed , and for a taken branch two stall cycles are needed)  What is the performance comparison between P1 and P2?   1. **CPI of P2 is less than the CPI of P1,Hence P2 is faster** 2. CPI of P1 is less than the CPI of P2,Hence P1 is faster 3. CPI of P2 is equal than the CPI of P1,hence P1 and P2 are equal. 4. CPI of P2 is greater than the CPI of P1,Hence P1 is faster   **Explanation**  **CPI of P1 = 1 + 0.1 ∗ 1 + 0.15 ∗ 2 = 1.4**  **CPI of P2 = 1 + 0.2 ∗ 1 + 0.05 ∗ 2 = 1.3**  **The CPI of P2 is less than the CPI of P1. Hence, P2 is faster** | | | |  |
| **10.** | **Pipelining Hazards :**  **Branch Prediction:**  **Unit-2** | | Explain the Branch History Table that is inevitable to predict in 1st stage of pipeline processor. | | | |  |
| **10** |  | | We want to perform the operation Y \* 253, Register R1 holds the value of Y and Register R2 holds the value 253. We expect register R1 to hold the result of the operation. Write ARM assembly program to perform the operation, without using the MUL or MLA instruction. Try to use as few instructions as possible. | | | |  |
| **11.** |  | | ;IF(R0==R1)R2++ ELSE R3--; USE CONDIIONAL EXECUTION and choose the correct code snippet in ARM-ISA.  **Answer:**  **MOV R0,#5**  **MOV R1,#5**  **CMP R0,R1**  **ADDEQ R2,R2,#1**  **SUBNE R3,R3,#1**  **SWI 0X011** | | | |  |
| **12** |  | | Write a program to convert the following HLL to assembly language, assume  R1 = 0x9, R2 = 0x6 and R3 = 0x5  If R1>R2 Then R3=R3-5 | | | |  |
| **13** |  | | Write a program in ARM7TDMI-ISA to check the parity of given 32 bit number using function subprogram PARITYCHECK. Display appropriate messages as ODD PARITY or EVEN PARITY number.  .DATA  ODD: .ASCIZ "ODD PARITY NUMBER"  EVEN: .ASCIZ "EVEN PARITY NUMBER"  A: .WORD 0X12345678    .TEXT  LDR R2,=A  LDR R3,[R2]  MOV R6,#0 ; COUNT REGISTER  MOV R4,#0 ; COUNT NUMBER OF 1S.    L2: TST R3,#1  BNE L1  LOOP:MOV R3,R3,LSR #1  ADD R6,R6,#1  CMP R6,#32  BNE L2  TST R4,#1  BEQ L3  LDR R0,=ODD  SWI 0X02  B EXIT  L3: LDR R0,=EVEN  SWI 0X02    EXIT:SWI 0X011    L1:ADD R4,R4,#1  B LOOP    .END | | | |  |
| **14** |  | | Let register R0=0XFFFFFFFF and R13=0x008010. Show the content of stack and update on stack pointer when following instructions are executed.  **i.STR R0,[SP,#-8]**  **ii. STR R0,[SP,#-8]!** | | | |  |