Computer Science 605.611

Problem Set 13

Module 13 provides descriptions of the MIPS, Sparc V8, IA-32, and ARM processor architectures. Your answers for this problem set should be based on those descriptions.

1. As you know, the MIPS processor described in this course employs a 5-stage pipeline. The PC is incremented by 4 in stage 1 each time that an instruction is fetched. This incremented PC value is passed on to the following pipeline stages via the MIPS pipeline registers. When a MIPS beq instruction is in the execute stage, the low 16 bits in the instruction are shifted left 2 bits and sign extended into an equivalent 32-bit signed integer. This 32-bit signed integer is added to the PC value from the ID/EX pipeline register and the sum is used as the branch target address for the MIPS beq instruction.

The ARM7 processor employs a 3-stage pipeline (fetch, decode, execute). The ARM7 execute stage effectively combines the activity performed in the MIPS execute, memory and write-back stages. ARM7 CPU register 15 is used as the PC (program counter). This ARM7 PC register is incremented by 4 in the fetch stage each time that an instruction is fetched. The ARM7 beq instruction, contains a signed integer in its low 24 bits. This 24-bit signed integer is shifted left two bits and signed extended to produce a 32-bit signed integer. The 32-bit signed integer is added to the ARM PC register and the sum is used as the branch target address for the ARM beq instruction.

Assume that the MIPS processor and the ARM7 processor must each execute a beq instruction that resides within its memory at address 0x400B5678. Recall that addresses are just 32-bit unsigned integers.

a) (3) What is the highest address to which this MIPS beq instruction can branch?

Highest address =  **0x400D5678**

Highest 16 bit value = 0x7FFF, this then needs to be shifted left: 0x1FFFC, then we have to sign extend it to 32 bits 0x0001FFFC. To determine the branch instruction max we have to add 4 to the current address since its 4 stages in when it is actually ran, 0x400B5678 + 4 = 0x400B567C. Finally we add the two numbers together, 0x400B567C + 0x0001FFFC = 0x400D5678

b) (3) What is the highest address to which this ARM beq instruction can branch?

Highest address = **0x420B567C**

To determine this we start with the highest 24 bit signed int = 0x7FFFFF. Then we shift it left 2 bits, 0x1FFFFFC, sign extended it to 32 bits leaves us with 0x01FFFFFC. Then we can find the base address 0x400B5678 + 8 = 400B5680, so 0x400B5680 + 0x01FFFFFC = 0x420B567C

2. As described in module 13, ARM instructions are conditionally executed (based on various combinations of the N, V, Z and C condition bits). In general, the MIPS processor does not include or support the conditional execution of instructions. However, the MIPS processor does support a few conditional move instructions such as movz.

a) (3) Prior to executing the following two ARM instructions, registers R3, R4 and R5 contain the values 3, 4 and 5, respectively.

CMP R3, #0

MOVEQ R5,R4

Show the contents of ARM registers R3, R4 and R5 after the two instructions execute.

CMP R3, #0 compares R3 with 0. Since R3 is not 0, the zero flag will not set. Then MOVEQ checks if R5 and R4 are the same, since they are not it does nothing.

**R3 = 3, R4 = 4, R5 = 5**

b) (3) Show a single MIPS conditional move instruction that produces the same effect on the MIPS registers $3, $4 and $5 as the effect produced by the above pair of ARM instructions on the ARM registers R3, R4 and R5. Assume that initially MIPS register $3 contains 3, register $4 contains 4 and register $5 contains 5.

**Movz $5, $4, $3**

This would perform the same as the ARM instructions since it does a conditional check if $3 is 0, and then only performs the move of $4 into $5 if it is true, in this case it is not.

3. a) (3) The IA-32 processor employs little endian memory storage order. An instruction on this processor writes the 32-bit pattern 0xFEEDBABE contained in the EAX register into memory at address 0x80084004. Show the hex contents of the 16-bit AX register after the halfword at address 0x80084005 is then read back into the AX register.

The contents of the AX = \_**0xBAED**\_\_

Stored into addresses 0x80084004 – 0x80084007 as 0xBE, 0xBA, 0xED, 0xFE. Then it reads back 0x80084005 and 0x80084006 leaving us with AX = BAED.

b) (3) The SparcV8 processor employs big endian memory storage order. Assume that register %o7 contains the pattern 0xC0FFEE41 and register %i5 contains the pattern 0x80084004.

Use hex to show the contents of register %o6 after the following two Sparc instructions are executed:

st %o7, [%i5]

ldsh [%i5+2],%o6

The instructions st and ldsh are the Sparc store word and load signed halfword instructions, respectively.

The contents of the low half of %o6 = 0x\_ **EE41**\_

st %o7, [%i5] loads content of reg %o7 into memory location pointed to by %i5, so 0x80084004 – 0x80084007 as 0xC0, 0xFF, 0xEE, 0x41. The ldsh [%i5+2],%o6 loads a signed halfword from memory address %i5+2 into %o6. %i5+2 = 0x80084006. Thus %o6 contains 0xEE41

4. Recall that an integer is just a whole number (a value with a zero fractional part).

a) (3) Indicate the largest positive odd whole number that can be represented exactly using the IA-32 processor’s internal 80-bit floating-point format described in module 13.

This would be 2^64 – 1, so **18,446,744,073,709,551,615**

b) (3) Indicate the largest positive odd whole number that can be represented exactly using the IEEE-754 64-bit double precision format employed by the Sparc-V8 processor.

This would be 2^52 – 1, so **4,503,599,627,370,495**

5. a) (5) Assume that a very simple function is called on the ARM processor using the ARM BL instruction. The function has no input arguments. Show a pair of ARM instructions within the function that will place the return address into register R2 and then return to the calling program.

**MOV R2, LR**

**BX LR**

b) (5) Assume that a very simple function is called on the Sparc V8 processor using the Sparc call instruction. The function has no input arguments. Show a pair of Sparc V8 instructions within the function that will place the return address into register %o2 and then return to the calling program.

**MOV %o7, %o2**

**JMPL %i7+8, %g0**

c) (5) Assume that a very simple function is called on the MIPS processor using the MIPS jal instruction. The function has no input arguments. Show a pair of MIPS instructions within the function that will place the return address into register $v0 and then return to the calling program.

**MOVE $v0, $ra**

**JR $ra**

d) (5) Assume that a very simple function is called on the IA-32 processor using the IA-32 call instruction. The function has no input arguments. Show a pair of IA-32 instructions within the function that will place the return address into register EAX and then return to the calling program.

**POP EAX**

**RET**

6. Answer the following questions for each of the MIPS, Sparc V8, IA-32, and ARM processors and explain your answer.

a) (3) Can the processor annul the instruction in the branch delay slot? “Annulling” an instruction means preventing it’s execution.

MIPS\_\_\_YES\_\_\_ --MIPS does have branch instructions that allow for annulling

Sparc V8\_\_YES\_\_ -- Sparc has bits meant specifically for annulling

IA-32\_\_\_NO\_\_\_\_ -- IA-32 does not use delay slots so it cannot annul

ARM\_\_\_NO\_\_\_ --ARM does not use branch delay slots

b) (3) Can the processor use an add instruction to add one of its CPU registers to the PC (program counter)?

MIPS\_\_YES\_\_ -- MIPS allows arithmetic options to be performed on the PC

Sparc V8\_\_\_ YES \_\_\_\_ --Sparc v8 Allows arithmetic operations to be performed on the PC

IA-32\_\_\_\_\_ NO \_\_\_\_\_ --IA-32 does not allow a direct addition to the PC (but can be worked around so kinda?)

ARM\_\_\_\_ YES \_\_\_\_\_\_ --ARM allows operations to be performed on the PC

7. The label “table” corresponds to address 0x04E37888 of a 32-bit word in a big endian memory. The 32-bit pattern contained in this memory word is 0xBABEFACE.

a) (5) Use hex to show the contents of register R2 on an ARM system after the following ARM instruction is executed.

ADR R2, table

This moves table into R2, so **R2 contains 0x04E37888**

b) (5) Use hex to show the contents of register $v0 on a MIPS system after the following MIPS instruction is executed.

la $v0, table

The contents of **$v0 would be 0x04E37888**, since la loads the address.

c) (5) Show a single MIPS instruction that has the same effect on the MIPS processor as the following ARM instruction has on the ARM processor:

MOV R15, R14

MOV R15, R14 moves the contents of R14 (link register) into R15 (which is the program counter). In mips we would most likely use the jump register instruction. We would do something like **jr $r14**

8. The ARM instructions STM (store multiple) and LDM (load multiple) described in module 13 allow multiple registers to be stored into or read from a block of contiguous memory words. These instructions can be used with the stack pointer register (R13) to push multiple registers onto the stack or pop multiple registers from the stack.

a) (5) The ARM instruction STMDB R13!, {R11, R2-R6} pushes the 6 registers R2, R3, R4, R5, R6 and R11 onto the stack. Show a series of no more than 7 MIPS true-op assembly language instructions that will accomplish the same task on the MIPS processor for the corresponding MIPS registers ($2, $3, $4, $5, $6 and $11).

The MIPS true op assembly language instructions would look like this:

Addi $sp, $sp, -24

Sw $11, 20($sp)

Sw $6, 16($sp)

Sw $5, 12($sp)

Sw $4, 8($sp)

Sw $3, 4($sp)

Sw $2, 0($sp)

b) (3) If ARM register R13 contains the 32-bit pattern 0xC0080000 prior to executing the instruction LDMIA R13!, {R11, R4, R6}, what value will register R13 contain after the instruction executes? Express your answer as an 8-digit hex number.

Lets start by understanding what LDMIA is doing, loading multiple with increment after. So R13 starts with 0xC0080000. We need to then increment that value by 12 bytes, (4 bytes \* 3 registers) so 0xC008000 + 0x0000000C = **0xC008000C**

9. Module 13 describes how the SparcV8 processor makes use of “register windows”. Recall that when one function calls another, the called function can execute the Sparc “save” instruction to slide the window so that a new set of registers can be accessed within the called function. The table below shows the names assigned to each of the 32 Sparc registers that can be accessed by a function or procedure. The 8 global registers (%g0 to % g7) are accessible to all functions. The remaining 24 registers are treated like logical registers, each of which must be mapped to one of the physical hardware registers. Recall that “out” registers in one window overlap with the “in” registers in the next adjacent register window. The final register window overlaps with the first or initial register window.

|  |  |
| --- | --- |
| Register names | Register numbers |
| %g0 - %g7 | %r0 - %r7 |
| %o0 - %o7 | %r8 - %r15 |
| %l0 - %l7 | %r16 - %r23 |
| %i0 - %i7 | %r24 - %r31 |

a) (3) What is the main advantage of using register windows?

The main advantage of using register windows is that it allows for a very optimized way of calling functions. This is because it overall reduces the need to access memory meaning that we don’t have to continuously save and restore registers.

b) (3) If the Sparc system allows only two register windows, what is the total number of physical CPU hardware registers required for the system?

There would only need to be 40 physical registers. This is because there would need to be 8 global registers, 2 sets of 8 local registers, and 2 sets of 8 in/out registers (the in becomes the out).

c) (3) The Sparc processor includes an instruction to slide the register window backwards to uncover the previous set of logical registers. What is this Sparc instruction?

The Sparc instruction that performs this is **restore**.

10. The ARM instruction MVN R2, #10 places an integer value into the ARM register R2. The MIPS instruction ori $2, $0, -12 also places an integer value into the MIPS register $2.

If the resulting bit patterns in register R2 and register $2 are interpreted as two’s complement integers, what decimal value does each bit pattern represent?

a) (3) The bit pattern in ARM register R2 represents decimal \_\_-11\_\_\_

10 in binary(32) is 0000 0000 0000 0000 0000 0000 0000 1010, then it is inverted 1111 1111 1111 1111 1111 1111 1111 0101. In 2’s compliment this is -11

b) (3) The bit pattern in MIPS register $2 represents decimal \_\_-12\_\_\_

There isn’t much that needs converting here since its or-ing against 0, its just -12

11. The MIPS processor described in this course uses 32-bit CPU registers. Integers that are 64 bits in size can be handled on the MIPS processor by using an even/odd pair of 32-bit registers. The even numbered register holds the leftmost 32 bits of the 64-bit integer and the next higher odd numbered register holds the rightmost 32 bits of the 64-bit integer.

Assume that MIPS registers $2 and $3 contain the binary equivalent of the decimal values 169482938 and 1516927626, respectively. Registers $4 and $5 contain the binary equivalent of the decimal values 455818059 and 1264282507, respectively.

The IA-32 processors with multi-media extensions include 64-bit MMX registers each of which can hold a 64-bit integer. Assume that the 64-bit IA-32 MM2 register contains the same 64-bit pattern as the 64-bit integer in MIPS registers $2 and $3. Also assume that 64-bit IA-32 register MM4 contains the same 64-bit pattern as the 64-bit integer in MIPS registers $4 and $5. The even numbered MIPS registers contain the high 32 bits of the 64-bit integer.

a) (3) Show, in hex, the contents of register MM2 after the IA-32 instruction

PADDB MM2, MM4 is executed.

contents of MM2 = **0x25545505A5C5E615**

PADDB is the byte-wise addition of the MMX registers. MM2 contains the values in $2 and $3 so 0x0A1A1ABA5A6A7A8A. MM4 contains the values in $4 and $5 so 0x1B2B3B4B4B5B6B8B. Then we need to add byte by byte and get 0x25545505A5C5E615

b) (3) Show, in hex, the contents of MIPS registers $2 and $3 after the following two MIPS instructions are executed:

addu $2, $2, $4

addu $3, $3, $5

$2 in hex = 0x25455605 $3 in hex = 0xA5C5E615.

Addu does unsigned addition, the first set would be adding $2 and $4 and storing that in $2 and then the second instruction would add $3 and $5 and store it in $3.

12. a) (3) Assume that MIPS registers $2 & $3 together contain one 64-bit integer and registers $4 & $5 together contain another 64-bit integer. The even numbered register holds the high 32 bits and the odd numbered register holds the low 32 bits. Show one or more MIPS true-op instructions that together produce the 64-bit double precision integer sum of these two 64-bit integers in the result register pair $6 & $7.

**Addu $7, $3, $5**

**Sltu $at, $7, $3**

**Addu $6, $2, $4**

**Addu $6, $6, $at**

b) (3) Show two ARM instructions that together place into the register pair R6 & R7 the 64-bit integer sum of the two 64-bit integers in the register pair R2 & R3 plus the 64-bit integer in the register pair R4 & R5. The even numbered register contains the high 32 bits and the odd numbered register contains the low 32 bits of the numbers.

The two ARM instructions are as follows:

**ADDS R6, R2, R4**

**ADC R7, R3, R5**