Computer Architecture

Fall, 2025
Week 4
2025.09.22

組別：_____ 簽名：_____

[group 1]

1. **True or False: Addressing for 32-bit addresses**

   a. MIPS branch instructions (beq, bne) can encode a full 32-bit target address directly within the instruction, since the immediate field is sign-extended to 32 bits.

   b. The 16-bit immediate in a branch instruction is interpreted in bytes, so the branch offset range is limited to ±32 KB in the address space.

   c. MIPS jump (j, jal) instructions achieve 32-bit target addresses by concatenating the 26-bit target field with the upper 4 bits of the current PC and appending two zero bits, forming a word-aligned address.

   d. The reason MIPS uses PC-relative addressing for branches and pseudodirect addressing for jumps, rather than storing full 32-bit addresses, is that the 32-bit fixed instruction format cannot accommodate both the opcode and an entire 32-bit address at once..

Ans:

   a. False. MIPS branch instructions do not directly encode full 32-bit addresses; they instead rely on PC-relative addressing with a 16-bit signed offset. This offset is sign-extended, but its purpose is to specify a relative distance from PC+4, not to encode an absolute address.
   b. False. Branch offset is given in words, not bytes. This means the 16-bit immediate is shifted left by two bits, yielding a range of ±2^15 words, or ±2^17 bytes (about ±128 KB), which is significantly larger than ±32 KB.
   c. True. The pseudodirect jump mechanism in MIPS combines the 26-bit target from the instruction with the high 4 bits of the PC and appends two zeros.
   d. True. With only 32 bits available per instruction, it is impossible to hold both a 6-bit opcode and a full 32-bit address field.

[group 4]

2. Question:

Why are there beq and bne instructions for == and !=, but no blt (<) or bge (>=) instructions?

Ans:

hardware for blt (<) and bge (>=) are slower (more complex) than beq (==) and bne (!=), beq and bne are the common case. And it's a good design compromise.

[group 5]

3. Question:

In ARM, the top 4 bits of each instruction specify a condition code. What is the benefit of this feature?

Ans:

Conditional execution allows short conditional statements to be executed without using branch instructions. (This reduces pipeline flushes and branch misprediction penalties, improving performance.)

[group 3]

4. Question:

sum:

```
addi $sp, $sp, -8
sw   $ra, 4($sp)
sw   $a0, 0($sp)
add  $v0, $a0, $a1    # v0 = a0 + a1
lw   $a0, 0($sp)
lw   $ra, 4($sp)
addi $sp, $sp, 8
jr   $ra
```

If the function is called as sum(7, 5), what is the return value stored in $v0?

Why does the function begin with the instruction addi $sp, $sp, -8?

Ans:

$v0 = 12

To allocate 8 bytes of stack space for saving $ra and $a0, preventing their values from being overwritten during the function call.

[group 7]

5. Question:

Assume a beq $s1, $s2, Exit instruction is at memory address 80012, and the Exit label is at address 80024. Calculate the immediate value for the beq instruction's 16-bit address field.

Ans:

To find the immediate value, we use the PC-relative address formula:
Target Address = (PC + 4) + (immediate × 4)
Plugging in the given values:
80024 = (80012 + 4) + (immediate × 4)
80024 - 80016 = immediate × 4
8 = immediate × 4
Therefore, the immediate value is 2.

[group 8]

6. Question:

What is the difference between a leaf procedure and a non-leaf procedure in MIPS, and how does the use of the stack differ between them

Ans:

- Leaf procedure:
  - A procedure that does not call another procedure.
  - Only needs to save its own local variables (e.g., $s0) and temporaries if required.

  Example: leaf_example saves $s0 on the stack, performs calculations, restores $s0, and returns with jr $ra.

- Non-leaf procedure:
  - A procedure that calls another procedure (e.g., recursive functions).
  - Must save not only local variables but also the return address ($ra) and any arguments or temporaries that are still needed after the call.

Example: fact (factorial) saves $ra and $a0 on the stack before making a recursive call, restores them afterward, and then multiplies the result.

[group 9]

7. Question:

We use J-format instruction to implement jump addressing. However, the op code occupies 6 bits, so there are only 26 bits for the target address. How can we specify a 32-bit memory address with only 26 bits? Please explain.

Ans:

First, the last two bits are always 00 because each word has 4 bytes. After that, the remaining 4 bits (highest order) are taken from the 4 highest order bits of the program counter, then we have 4 + 26 + 2 = 32 bits to specify an address.
However, we should avoid placing a program across an address boundary of 256MB, since the memory is divided into $2^4 = 16$ sections and we can only access to one of them simultaneously.

[group 11]

8. Question:

Caller is in charge of placing parameters in registers, transferring control to procedure. Does it acquire storage for procedure? Why or why not.

Ans:

No. Caller doesn’t acquire the storage for the procedure. Callee does.

[group 12]

9. What are the five addressing modes of MIPS?

Ans:

1.Immediate Addressing Mode
2.Register Addressing Mode
3.Base or Displacement Addressing Mode
4.PC-relative Addressing Mode
5.Pseudodirect Addressing Mode

[group 13]

10. Question:

Turn this C code into MIPS instruction (Assume the input n is n >= 0)

```
int pow2(int n){
    if ( n < 1)      return 1;
    else return 2 * pow2(n-1);
}
```

Ans:

```
Pow2:
    addi   $sp $sp -8
    sw     $ra 4($sp)
    sw     $a0 0($sp)
    slti   $t0 $a0 1
    beq    $t0 $zero L1
    addi   $v0 $a0 1
    addi   $sp $sp 8
    jr     $ra
L1:
    addi   $a0 $a0 -1
    jal    pow2
    lw     $a0 0($sp)
    lw     $ra 4($sp)
    addi   $t0 $t0 2
    addi   $sp $sp 8
    mul    $v0 $t0 $v0
    jr     $ra
```

[group 14]

11. Question:

Let the value of the program counter be 0xCF01FC00. What is the target address of the instruction, j 0x20.

Ans:

0xC0000080