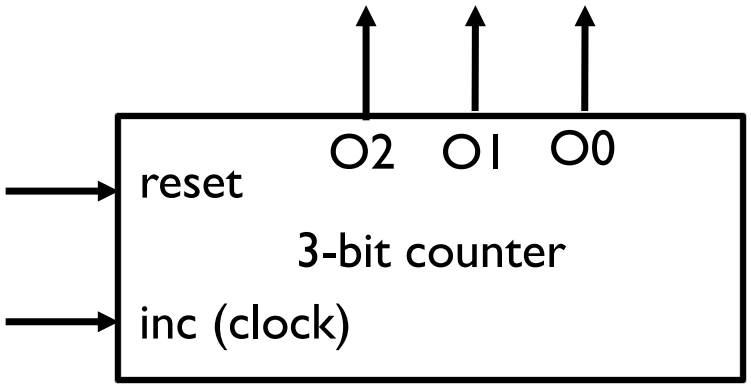
Computer Systems
#7. Instructions (Logical and Branch)

Keiji Kimura <keiji@waseda.jp>

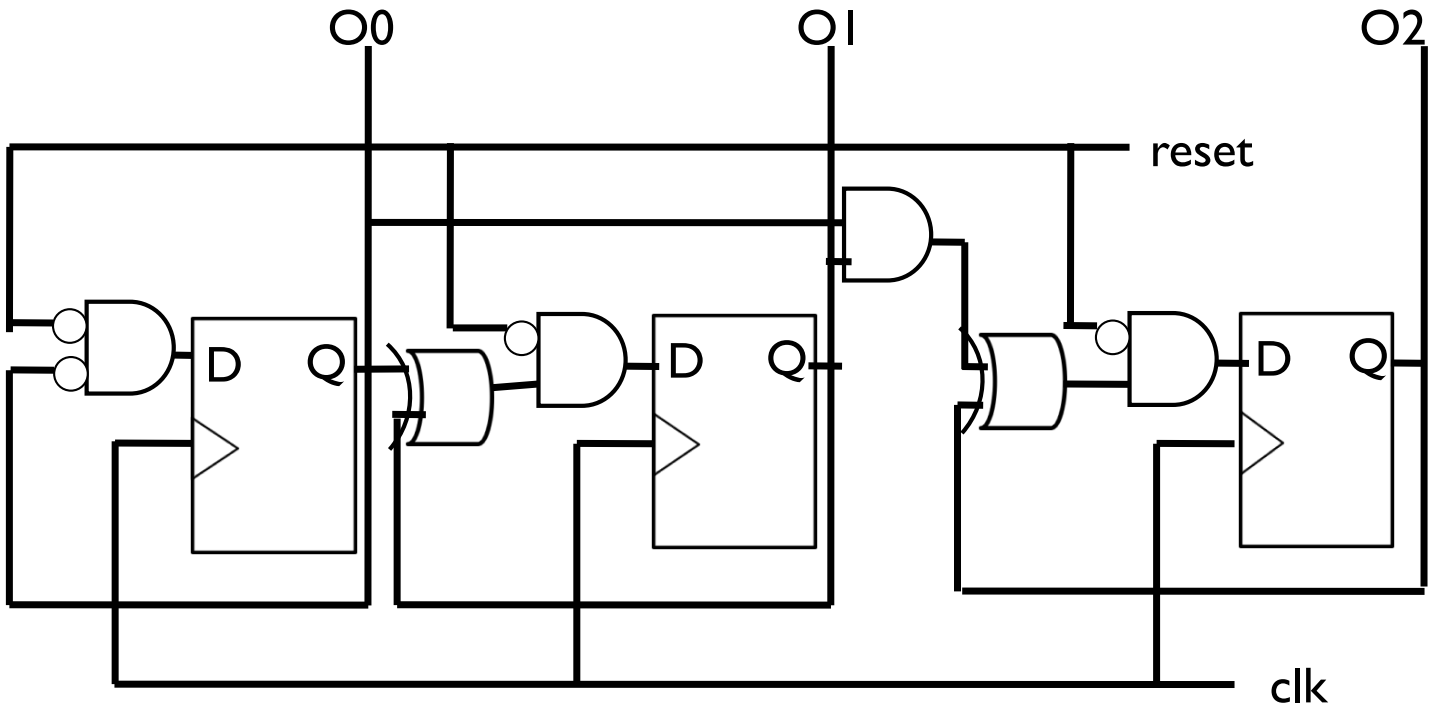
Ans. for Exercise (1)

- Construct a 3-bit counter using three D flip-flops and a selection of gates. The inputs should consists of a signal that resets the counter to 0, called reset, and a signal to increment the counter called inc. The outputs should be the value of the counter. When the counter has value 7 and is incremented, it should wrap around and become 0.



Ans. for Exercise (1) cont.

	O2	O1	O0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
0	0	0	0



Each bit depends on
its own previous state
and previous lower bits.

Assignment

- 2nd assignment will be placed on CourseN@vi.
- Dead line: June 14th.

Exercise (1)

- Show the corresponding MIPS assembly code. Assume that the variables f, g, h, i, and j are assigned to register $s0, $s1, $s2, $s3, and $s4, respectively. Assume that the base address of the arrays A and B are in registers $s6 and $s7, respectively. Note that the size of each variable is 4 bytes.
  - f = -g – A[4];
  - B[8] = A[i-j];

Immediate operands

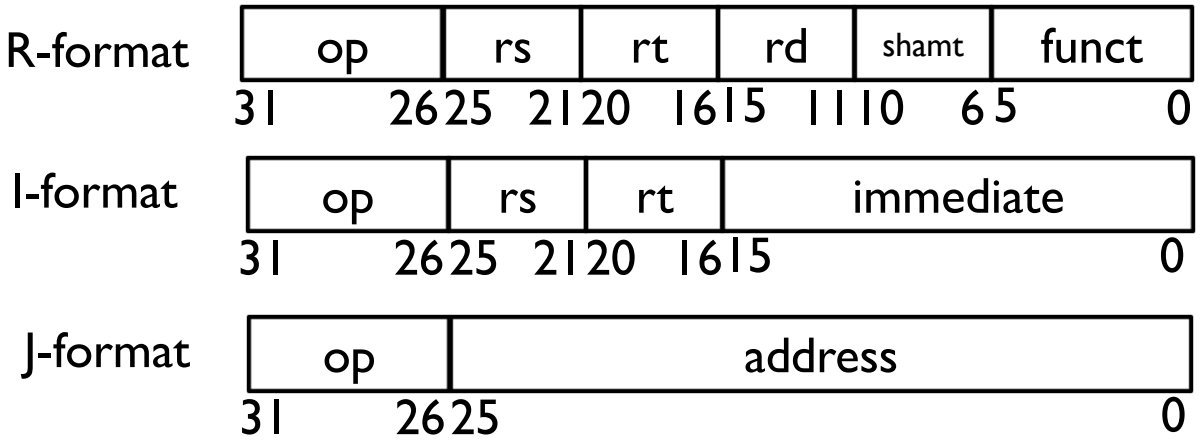
- An instruction sometimes requires an immediate value instead of a register.
  - Just increment one-word memory location
- addi $s3, $s3, 4 # $s3=$s3+4
  - The size of an immediate operand is restricted (as shown later).
- loading 32bit immediate value
  - Use lui (load upper immedite) instruction
  - ex. copy 0x12345678 to $s0
    lui $s0, 0x1234
    ori $s0, $s0, 0x5678

Representing Instructions

- Each instruction of a computer is represented as a number.
  - Machine language (cf. assembly language)
  - Especially, each MIPS instruction is represented as a 32-bit number.
    - cf. ARM: 16-bit, Intel x86: variable length
- ex) add $t0, $s1, $s2
  - This instruction contains 'operation type', 'destination register', '1st source register', and '2nd source register'.
  - Representation in binary numbers
    000000 10001 10010 01000 0000 100000
    - hint: $t0->$r8, $s1->$r17, $s2->$r18

Format

- MIPS has three types of formats
  - R-format, I-format, J-format
  - These formats can be distinguished by 'op' field



- op (opcode): Basic operation of the instruction
- rs: the 1st source operand
- rt: the 2nd source operand
- rd: the register destination operand
- shamt: shift amount (for shift instructions)
- funct: function. the specific variant of the operation in the op field

ex) MIPS instruction encoding

Instruction	Format	op	rs	rt	rd	shamt	funct	immediate
add	R	0	reg	reg	reg	0	32(10)	n.a.
sub	R	0	reg	reg	reg	0	34(10)	n.a
add immediate (addi)	I	8(10)	reg	reg	n.a.	n.a.	n.a.	constant
lw	I	35(10)	reg	reg	n.a.	n.a.	n.a.	address
sw	I	43(10)	reg	reg	n.a.	n.a.	n.a.	address

Instruction width and Immediate operands

- loading 32bit immediate value
  - Use lui (load upper immedite) instruction
  - ex. copy 0x12345678 to $s0
    lui $s0, 0x1234
    ori $s0, $s0, 0x5678

Logical Operation

- Logical operations in MIPS
  - sll: shift left
  - srl: shift right
  - and, andi: bit-by-bit AND
  - or, ori: bit-by-bit OR
  - nor: bit-by-bit NOT
- ex)
  - $t1 = $t1*4
    sll $t1, $t1, 2
  - $t1= $t1/8
    srl $t1, $t1, 3
  - exploit lower 8bits from $t1 into $t2
    andi $t2, $t1, 0xff

Exercise (2)

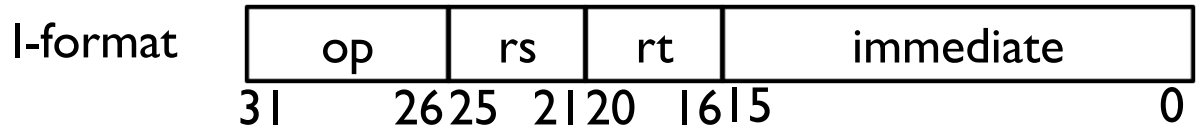
- Translate following expression into MIPS assembly language, then encode into machine code under the assumption that $t1 has the base of the array A and $s2 corresponds to h:
  - A[300] = h+A[300]; /* All variables are 32bit integer */
- Exploit rt field into $t1 when $s0 has an instruction code.

branch

- Conditional branch
  - Based on the condition at that time, jump to other instructions
- MIPS has two conditional branch instructions
  - beq register1, register2, L1
    - Jump to L1 if register1 equals to register2
  - bne register1, register2, L1
    - Jump to L1 if register 1 doesn't equal to register2

PC relative addressing

- PC (program counter)
  - contains the address of the current instruction.
- beq and bne are I-format



  - immediate has only 16bits width
  - Not enough for representing an instruction address (32bits)
- PC relative addressing
  - Target address is calculated as (PC+4)+immediate*4.
    - immediate is signed value.
    - Why (PC+4)? Why immediate*4?
- j (jump instruction)
  - J-format (26bits)
  - "address" represents target address (in word address)

MIPS Addressing Mode Summary

- Addressing modes
  - multiple forms of addressing
- MIPS addressing modes
  - Immediate addressing
    - The operand is a constant within the instruction itself. (I-format)
  - Register addressing
    - The operand is a register. (R-format)
  - Base or displacement addressing
    - The operand is at the memory location whose address is the sum of a register and a constant in the instruction. (lw/sw)
  - PC-relative addressing
    - The branch address is the sum of the PC and a constant in the instruction.
  - Pseudo direct addressing
    - The jump address is the 26bits of the instruction concatenated with the upper bits of the PC. (J-format)

if-then-else in MIPS assembly

- ex.) if (i==j) f = g+h; else f=g-h;
- Five variables f through j correspond to the five registers $s0 through $s4
- ```
      bne $s3,$s4, Else # if
      add $s0, $s1, $s2 # then part
      j Exit
  Else:
       sub $s0, $s1, $s2 # else part
  Exit:
  ```
  ```

Loops in MIPS assembly

- Important operation for iterative computations
- ex) while (save[i] == k) i++;
- i and k correspond to register $s3 and $s5 and the base of the array save is in $s6
- Loop:

```
    sll $t1, $s3, 2  # i*4 for save[i] (32bit data)
    add $t1, $t1, $s6
    lw $t0, 0($t1)
    bne $t0, $s5, Exit # (save[i] != k)?
    addi $s3, $s3, 1
    j  Loop
Exit:
```

Comparison

- beq and bne are for evaluating equality.
- How about less than and larger than?
- slt: set on less than
  - slt $t0, $s3, $s4 # $t0 = 1 if $s3 < $s4
- Isti
  - slti $t0, $s3, 10 # $t0 = 1 if $s3 < 10
- sltu/sltiu
  - for unsigned integers
- How to represent other relative conditions?
  - less than or equal, greater than, greater than or equal
- Why not blt?

Exercise (3)

- Show assembly code for the following loop under the assumption i corresponds to register $s3 and the base of the array save is in $s6.
  - for (i = 0; i < 10; i++) save[i] = 0;
- Show assembly code for the following statements under the assumption f through j correspond to the five registers $s0 through $s4.
  - if (i<=j) f = g+h; else f=g-h;
- Suppose register $s0 has the hexadecimal number 0xffffffff and the register $s1 has 1. What are the values of register $t0 and $t1 after the following instructions?
  - slt $t0, $s0, $s1
    sltu $1, $s0, $s1