HY225 Lecture 05: Instruction Formats

Dimitrios S. Nikolopoulos

University of Crete and FORTH-ICS

March 12, 2011

Instruction format

- Format specifies encoding of an assembly instruction in bits
- MIPS executes fixed-size, 32-bit instructions
- Groups of bits called fields describe instruction elements
  - Operation (addition, substraction, and, or, ...)
  - Registers read or written
  - Immediate operands
- Three instruction formats
  - R-format: Instruction reads values of two registers, performs operation on them, writes result in third register
  - I-format: Instruction reads one register and one immediate constant, performsn operation and writes result in second register

MIPS instruction set

- Register numbering
  - **$t0–$t7** are registers 8–15
  - **$t8–$t9** are registers 24–25
  - **$s0–$s7** are registers 16–23

R-format instructions

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- Instruction fields:
  - **op** : operation code (opcode)
  - **rs** : first source register number
  - **rt** : second source register number
  - **rd** : destination register number
  - **shamt** : shift amount (assume 00000, unless we execute a shift instruction)
  - **funct** : extends opcode

R-Format Example

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

add $t0, $s1, $s2

special	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

I-Format instructions

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

- Used in arithmetic instructions with immediate operands and load/store instructions
- Constant is
  - An integer in the range $-2^{15} \ldots 2^{15} - 1$
  - A signed offset in bytes from an address pointed to by a base register
  - A signed offset in words from the current PC, to jump to on a conditional branch
- **$rt** is
  - second source register (word, or half-word, or byte to be stored) for **store** instructions
  - second source register for **branch** instructions
  - destination register for all other I-format instructions

Shift instructions

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- **shamt** : specifies how many bit positions to shift
- **function** : distinguishes between logical left-right shifts, arithmetic right shifts, shifts using registers as shift amount, shifts using constants as shift amount
- Source register **$rs** unused if shift instruction uses a constant (**sll, srl, sra**)
- Source register **$rs** used as shift amount otherwise (**sllv, srlv, srav**)

Branch encoding

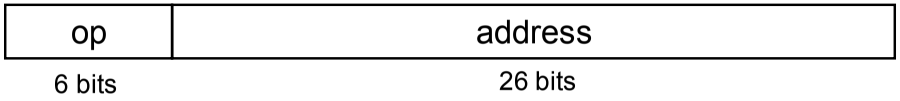
op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

- Branch needs to specify two source registers for comparison **($rs, $rt)** and target address
- Target address given by multiplying 16-bit constant field by 4
  - Constant field treated as signed integer
  - Signed integer encodes how many words ahead/behind current PC the branch target lies
- Branch addressing mode is called PC-relative addressing
  - **Branch target = PC + offset $\times$ 4**

Branch encoding in MIPS in more detail

- Real MIPS processors calcuate branh target as:
  - **Branch target = PC + offset $\times$ 4 + 4**
- The processor allows one instruction following the branch (at branch_PC + 4) to execute unconditionally, i.e. regardless of the branch being taken or not taken
- The instruction slot of the unconditionally executed instruction is called a delay slot
- We will ignore delay slots for now, will revisit when implementing the datapath

Jump addressing



- Jump (**j, jal**) targets are labeled locations, anywhere in the text segment
  - Need to encode full 32-bit address in instruction
  - Only 26 bits available though, since op-code needs at least 6 bits
  - MIPS uses pseudo-direct jump as:
    - 28 LS bits of target address = *address* $\times$ 4
    - 4 MS bits of target address same as current PC

Branch addressing example

```
Loop: sll  $t1, $s3, 2
      add  $t1, $t1, $s6
      lw   $t0, 0($t1)
      bne  $t0, $s5, Exit
      addi $s3, $s3, 1
      j    Loop
Exit: ...
```

- Assume code starts at **0x00408000**
- All fields in hex

Address	op	rs	rt	rd	shamt	funct
32-bit	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
0x00408000	0x0	0x0	0x13	0x9	0x2	0x0
0x00408004	0x0	0x9	0x16	0x9	0x0	0x20
0x00408008	0x23	0x9	0x8	0x0		
0x0040800C	0x5	0x8	0x15	0x3		
0x00408010	0x8	0x13	0x13	0x1		
0x00408014	0x2	0x0102000				

Branch addressing example in binary

```
Loop: sll  $t1, $s3, 2
      add  $t1, $t1, $s6
      lw   $t0, 0($t1)
      bne  $t0, $s5, Exit
      addi $s3, $s3, 1
      j    Loop
Exit: ...
```

- Assume code starts at **0x00408000**
- All fields in binary (instruction code in hex)

Address	Instruction encoding
0x00408000	0000 0000 0001 0011 0100 1000 1000 0000 (0x00134880)
0x00408004	0000 0001 0011 0110 0100 1000 0010 0000 (0x01364820)
0x00408008	1000 1101 0010 1000 0000 0000 0000 0000 (0x8d280000)
0x0040800C	0001 0101 0001 0101 0000 0000 0000 0011 (0x15150003)
0x00408010	0010 0010 0111 0011 0000 0000 0000 0001 (0x22730001)
0x00408014	0000 1000 0001 0000 0010 0000 0000 0000 (0x08102000)

MIPS addressing modes