5118007-02 Computer Architecture

Ch. 2 Instructions: Language of the Computer

12 Mar 2024

Shin Hong

Machine Instructions

- Instructions are the words of a computer's language
- Popular Instruction Sets
  - MIPS
  - ARMv7 and ARMv8
  - Intel x86
  - RISC-V
- MIPS assembly language
  - operation: arithmetic, data transfer, logical, branching, jump op.
  - operand: registers, memory locations, constants

Operations of Computer Hardware

```
add a, b, c  # a = b + c
sub d, e, f  # d = e - f
```

- Each arithmetic instruction performs only one operation and has exactly three operands
  - what if we want to add four values--`b`, `c`, `d`, and `e--`and store the result to a?
  - what about `f = (g + h) - (i +j )`?

Operands of Computer Hardware (1/3)

- Registers
  - a limited number of memory locations in a processor
  - a MIPS32 arch typically has 32 registers each of which is 32-bits
  - denoted as `$s0`, `$s1`, ..., `$s7, $t0,..$t9, $gp, $sp, $fp, $ra,` etc.
  - arithmetic instructions perform on registers only
- Memory
  - a memory is a large, single-dimensional byte array with the address acting as the index
  - load : copies data from memory to a register
  - store: copies data from a register to memory

Operands of Computer Hardware (2/4)

- Load Word Instruction
  - a single memory address contains a byte (i.e., 8-bits)
  - a MIPS32 processor can bring 4 bytes (i.e., 32-bits) at once from memory to a 32-bit register
  - a `lw` instruction loads 4 bytes consecutively aligned in memory to a register once the given address is a multiples of 4
    - alignment restriction
    - refer a word as the leftmost memory address (i.e., “big end”)

Operands of Computer Hardware (3/4)

```
int A[100] ;
A[12] = h + A[8] ;

lw $t0, 32($s3)   # assume $s has the base of A
add $t0, $s2, $t0 # assume $s2 represents h
sw $t0, 48($s3)
```

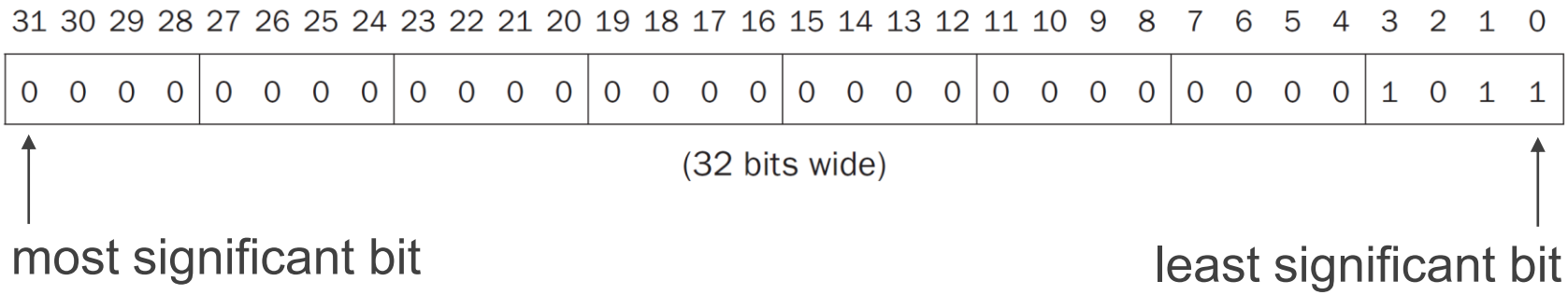
Operands of Computer Hardware (4/4)

- Constant
  - loading a constant stored in memory
    - Ex. `lw $t0, AddrConstant4($s1)`

      `add $s3, $s3, $t0  # s3 = 4`
  - having an immediate constant as an operand of an instruction
    - Ex. `addi $s3, $s3, 4  # s3 = 4`
  - referring the zero register $zero

Binary Number Representation

- A word in MIPS32 is 32 bits long, thus we can represent 2^{32} different numbers (e.g., 0 to 2^{32}-1)
- MIPS arranges bits of a word from right to left
  - Ex. 1011_{two} in 32-bits



Representing Singed Numbers

- Approach 1. Sign + Magnitude
  - problem of having positive and negative zero
  - complication at arithmetic operations

- Approach 2. Two's complement
  - B(a) + B(-a) = B(0)
  - B(-a) = Inverse(a) + 1
  - All negative numbers have a 1 in the most significant bit (sign bit)
  - Ex. -10_{ten} in 8 bits

Example

What is the decimal value of this 32-bit two's complement number?

$$1111\ \ 1111\ \ 1111\ \ 1111\ \ 1111\ \ 1111\ \ 1111\ \ 1100_{two}$$

Substituting the number's bit values into the formula above:

$$\begin{aligned}
&(1 \times -2^{31}) + (1 \times 2^{30}) + (1 \times 2^{29}) + \ldots + (1 \times 2^{1}) + (0 \times 2^{1}) + (0 \times 2^{0}) \\
&= -2^{31} + 2^{30} + 2^{29} + \ldots + 2^{2} + 0 + 0 \\
&= -2{,}147{,}483{,}648_{ten} + 2{,}147{,}483{,}644_{ten} \\
&= -4_{ten}
\end{aligned}$$

What is the decimal value of this 64-bit two's complement number?

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1000_{two}$$

Sign Extension

- A signed load copies the sign bit repeatedly to fill the rest of the register when *n*-bits value is loaded to a *m*-bits register for $n < m$
- load byte (`lb`) treats the byte as a signed number, thus sign-extends to fill the 24 left-most bits
- load byte unsigned (`lbu`) works with unsigned integer and does not sign-extend even if the left-most bit of a loaded byte is 1

Example

Convert 16-bit binary versions of 2_{ten} and -2_{ten} to 32-bit binary numbers.

The 16-bit binary version of the number 2 is

$$0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$$

It is converted to a 32-bit number by making 16 copies of the value in the most significant bit (0) and placing that in the left-hand half of the word. The right half gets the old value:

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$$

Let's negate the 16-bit version of 2 using the earlier shortcut. Thus,

$$0000\ 0000\ 0000\ 0010_{two}$$

becomes

$$\begin{array}{r} 1111\ 1111\ 1111\ 1101_{two} \\ +\qquad\qquad\qquad\quad 1_{two} \\ \hline = 1111\ 1111\ 1111\ 1110_{two} \end{array}$$

Creating a 32-bit version of the negative number means copying the sign bit 16 times and placing it on the left:

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -2_{ten}$$

Representing Instructions (1/2)

- A machine instruction is represented as a word-size number
  - a 32-bit number with MIPS32
  - assembly code vs. machine code
- The instruction format defines as a sequence of fields (i.e., layout)
  - R-format instruction
    - opcode
    - rs : the first register source operand
    - rt : the second register source operand
    - rd : the destination register
    - shamt : shift amount
    - funct : function code

Representing Instructions (1/2)

- The instruction format defines as a sequence of fields (con't)
  - I-format instruction
    - opcode
    - rs : the first register source operand
    - rt : the second register source operand
    - constant or address


```
lw   $t0,32($s3)   # Temporary reg $t0 gets A[8]
```

Register Name and Call Convention

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

MIPS Instruction Encoding

Instruction	Format	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	32_{ten}	n.a.
sub (subtract)	R	0	reg	reg	reg	0	34_{ten}	n.a.
add immediate	I	8_{ten}	reg	reg	n.a.	n.a.	n.a.	constant
lw (load word)	I	35_{ten}	reg	reg	n.a.	n.a.	n.a.	address
sw (store word)	I	43_{ten}	reg	reg	n.a.	n.a.	n.a.	address

- Example

```
int A[1000]  ;

A[300] = h + A[300] ;
```

```
lw  $t0, 1200($t1)
add $t0, $s2, $t0   #s2 is h
sw  $t0, 1200($t1)
```

Op	rs	rt	rd	address/ shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		