Tutorial 01

- Binary Intro
- Two's Compliment
- Machine Language (MIPS)

Binary

- Binary = Way data is represented in machines
  ↳ 2 digits : 0's & 1's
  ↳ has many interpretations!

eg: How could we interpret 1000 ?

soln:

① 8 (4-bit #)
② -8 (signed 4-bit #, two's compliment)
③ [True, False, False, False] (array of bools)
④ "backspace" char (ASCII code 8)

☆ bits can mean anything, interpretation is important!

- How can we convert binary to unsigned decimal? $n \geq 0$
  ↳ unsigned ints in binary are done with a positional number system representing powers of 2
  ↳ The decimal value = binary sum of 2's

eg: Convert 11010 to decimal

soln:

$$\begin{array}{ccccc} 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 1 & 1 & 0 & 1 & 0 \end{array} = 2^4 \cdot 1 + 2^3 \cdot 1 + \cancel{2^2 \cdot 0} + 2^1 \cdot 1 + \cancel{2^0 \cdot 0}$$

$$= 2^4 + 2^3 + 2^1 = 26$$

<u>In general</u>: A n-bit # is represented by the bits (b)

$b_{n-1} b_{n-2} \dots b_2 b_1 b_0$ $(b \in \{0,1\})$

has decimal value:

$2^{n-1} \cdot b_{n-1} + 2^{n-2} \cdot b_{n-2} + \dots + 2^2 \cdot b_2 + 2^1 \cdot b_1 + 2^0 \cdot b_0$

- How can we convert decimal to binary?

↳ Idea 1: Repeatedly subtract by large powers of 2 ≤ the decimal iteratively until the remainder = 0

<u>eg</u>: Convert 23 to binary

$23 - \underline{16} = 7$ 2^4 largest power of 2 ≤ 23

$7 - 4 = 3$ 2^2 " " " " ≤ 7

$3 - 2 = 1$ 2^1 " " " " ≤ 3

$1 - 1 = 0$ 2^0 " " " " ≤ 1

Thus: $23 = 2^4 + 2^2 + 2^1 + 2^0$

$= 2^4 \cdot \underline{1} + 2^3 \cdot \underline{0} + 2^2 \cdot \underline{1} + 2^1 \cdot \underline{1} + 2^0 \cdot \underline{1}$

$= 10111$

↳ Idea 2: Repeatedly divide by 2, read remainders from bottom to top

<u>eg</u>: Convert 23 to binary

$23 / 2 = 11$ remainder 1 2^0

$11 / 2 = 5$ remainder 1

$5 / 2 = 2$ " 1

$2 / 2 = 1$ " 0

$1 / 2 = 0$ " 1 2^4

∴ $23 = 10111$ in binary

eg: Convert 01101001 into decimal

Soln:

6 5 3 0

$01101001 = 2^6 + 2^5 + 2^3 + 2^0 = 105$

eg: Convert 35 to an 8-bit binary

Soln:

35/2 = 17	rem	1	2^0
17/2 = 8	"	1	
8/2 = 4	"	0	
4/2 = 2	"	0	
2/2 = 1	"	0	
1/2 = 0	"	1	2^5

Thus 35 = 100011 ← 6-bits? Pad with 0's !

= 00100011

eg: 216 to 8-bit binary

216/2 = 108	rem	0
108/2 = 54	"	0
54/2 = 27	"	0
27/2 = 13	"	1
13/2 = 6	"	1
6/2 = 3	"	0
3/2 = 1	"	1
1/2 = 0	"	1

216 = 11011000

Two's Compliment

- lets us represent positive & negative #'s in binary
  - ↳ for a n-bit binary $b_{n-1} b_{n-2} \ldots b_2 b_1 b_0$

$\begin{cases} \text{If the } 1^{st} \text{ bit } (b_{n-1}) \text{ is } 0, \text{ \# is positive} \\ \text{" " " " " " } 1, \text{ \# is negative} \end{cases}$

$\longrightarrow \underbrace{b_{n-1}}_{\text{sign: } 1=- ,\ 0=+} \underbrace{b_{n-2} \ldots b_2 b_1 b_0}_{\text{unsigned representation}}$

- Range of values with n-bits:
  - ↳ In unsigned binary: 0 to $2^n - 1$
  - ↳ In 2's compliment: -2^{n-1} to $2^{n-1} - 1$

1st bit reserved for ±

- Converting from decimal to 2's compliment
  - ↳ if the number ≥ 0, convert like an unsigned int

eg: 7 in 4-bit binary = 0111

  - ↳ if the number is < 0, then:

① Flip all bits in its positive binary representation
② Add 1

eg: −7 in 4-bit binary:

① $|-7| = 7 = 0111$ flip: all 0=1 & 1=0

1000 ← "1's compliment"

②
$$\begin{array}{r} 1000 \\ +\,0001 \\ \hline 1001 \end{array} = -7$$

- Convert from 2's compliment to decimal:

Method 1) if $b_{n-1} = 1$: flip bits, add 1, negate decimal

↳ Even faster: flip bits to the left of the rightmost 1

Method 2) leftmost bit is −2's power

$b_{n-1} \underbrace{b_{n-2} \dots b_2 b_1 b_0}$ ← unsigned #

$= -2^{n-1} \cdot b_{n-1} + 2^{n-2} b_{n-2} + \dots + 2^2 b_2 + 2^1 b_1 + 2^0 b_0$

↑ $b_{n-1} = 1$, # is < 0

$= 0$, # is ≥ 0

eg: Convert 1 0 1 0 0 to decimal

↳ Method 1)

$$
\begin{array}{r}
0\,1\,0\,1\,1 \\
+\ 0\,0\,0\,0\,1 \\
\hline
0\,1\,1\,0\,0
\end{array}
$$

$01100 = 2^3 + 2^2 = 12 \longrightarrow -12$ $(\because b_4 = 1)$

↳ Method 2)

↙ # is < 0

$\underline{1}\ 0\ 1\ 0\ 0$

$= -2^4 + 2^2 = -16 + 4 = -12$

Machine Language

- CS241 : 32-bit MIPS
  - ↳ 4 bytes of instructions per word in RAM (MEM)
  - ↳ Registers hold 32-bits of information ($s, $t, $d)
    - → Act like variables: Store data for access & manipulation
    - → Special registers
      - $0 : Always contains 0, immutable
      - $31 : Return address of the program
      - $3, $29 & $30 are special by convention

      don't use for storage!
      - PC contains the address to the current instruction
      - IR contains the instruction to run from MEM[PC]
  - ↳ iii... = 2's compliment numbers
- ☆ Programs live in the same space in RAM as the data they operate on! ☆
  - ↳ PC does not know how to distinguish the two!
- Fetch-Execute Cycle Pseudocode:

```
PC = 0x00
while True do:              // loop until PC = $31
        IR = MEM[PC]        // Instruction at address PC
        PC = PC + 4         // Next instruction address
        execute command in IR
done
```

- Constant Values
  - ↳ To load constant values in registers, use the Load Immediate Skip instruction (lis $d) followed by a 2's compliment number

eg: Store value 42 into register 5

```
lis $5
.word 42
```

treats 32-bit word after lis $5 as a constant to load into $5

Machine Code Representation: (MIPS Reference Sheet)

opcode	$s	$t	$d		lis	
000000	00000	00000	00101	00000	010100	lis $5
000000	00000	00000	00000	00000	101010	.word 42

eg: load in the number 5 into $1 & 10 into $2, then store the sum of $1 & $2 into $4.

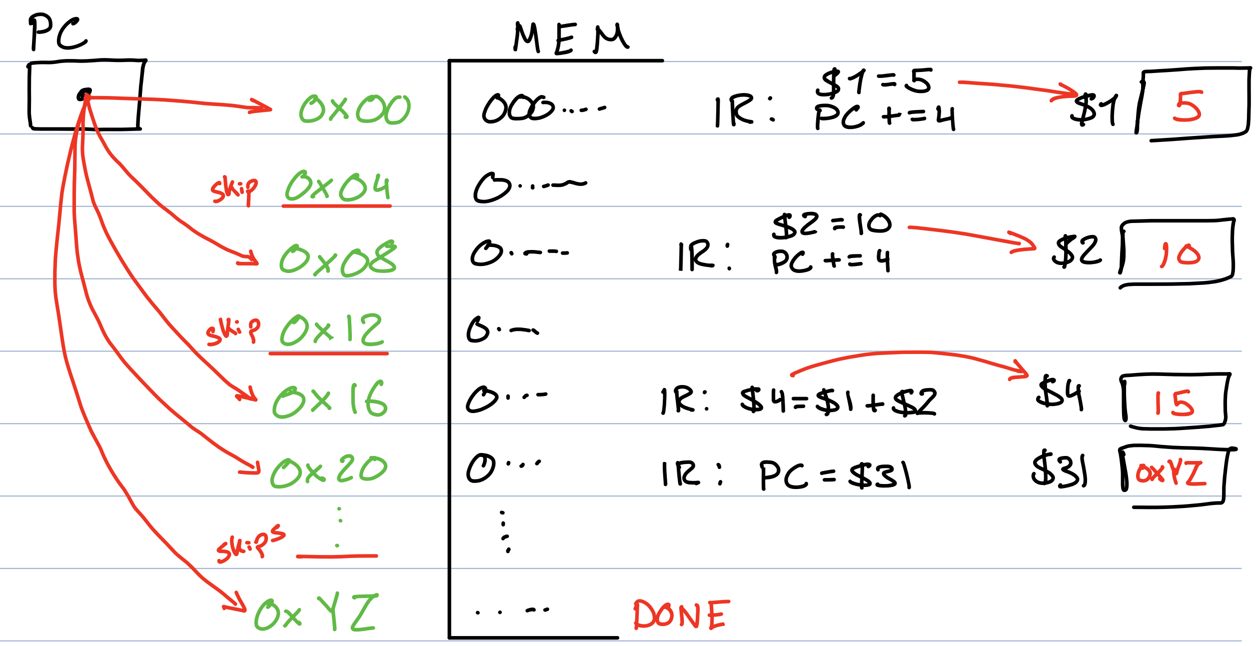
soln: (.word n ≡ 2's compliment binary of n)

```
lis $1
.word 5         //Stores value 5 in $1
lis $2
.word 10        //Stores value 10 in $2
add $4 $1 $2    // $4 = $1 + $2
jr $31          //Causes PC = $31, ends program
```

Converted into machine code:

address	opcode	$s	$t	$d		$n	
0x00	000000	00000	00000	00001	00000	010100	lis $1
0x04	000000	00000	00000	00000	00000	000101	.word 5
0x08	000000	00000	00000	00010	00000	010100	lis $2
0x12	000000	00000	00000	00000	00000	001010	.word 10
0x16	000000	00001	00010	00100	00000	100000	add $4 $1 $2
0x20	000000	11111	00000	00000	00000	001000	jr $31, DONE

Memory Diagram:



eg: load in $31*(2^{21})$ into $1 & 8 into $2, write a program that stops without using any jump operations

↳ Goal: Recreate the machine code to call PC=$31

↳ Recall: each word contains 4 bytes starting from 0x00

↳ Idea: Store Word will save a register value to a specific memory address (MEM[$s+i] = $t)

☆ what if MEM[$\underbrace{\$0 + PC + 4}_{\text{address of next instruction IR runs!}}$] = "PC = $31"

☆ Dynamically created instruction! ☆

soln:

0x00 lis $1

0x04 .word 31*2^21 000000 11111 00000 00000 00000 000000

0x08 lis $2

0x12 .word 8 000000 00000 00000 00000 00000 001000

0x16 add $\boxed{\$4}$ $1 $2 000000 11111 00000 00000 00000 001000

$\boxed{\$4 = jr\ \$31}$

0x20 MEM[$\underbrace{\$0 + 24}_{PC+4}$] = $4 jr $31 (ie: PC=$31)

↑ MEM address

Machine Code:

0x00	000000	00000	00000	00001	00000	010100	lis $1
0x04	000000	11111	00000	00000	00000	000000	.word $31*2^{21}$
0x08	000000	00000	00000	00010	00000	010100	lis $2
0x12	000000	00000	00000	00000	00000	001000	.word 8
0x16	000000	00001	00010	00100	00000	100000	add $4 = $1 + $2
0x20	101011	00000	00011	00000	00000	011000	MEM[$0+24] = $4
NEW 0x24	000000	11111	00000	00000	00000	001000	PC=$31 created!!!!

Memory Diagram