$\boxed{12}$ Computer Architecture

$\boxed{12.1}$ Binary Numbers

Binary Numbers: Base 2, ‘bit’ abbreviates ‘binary digit’.

bit ::= 0 | 1

Counting in binary: 0, 1, 10, 11, 100, 101, 110, 111, 1000, . . .

Binary Addition

11 11		$\longleftarrow$ carries
01100100	100_{10}	
00101110	46_{10}	
10010010	146_{10}	

$1 + 1 = 0$ with a carry; $\quad 1 + 1 + 1 = 1$ with a carry

$\boxed{12.2}$ Conversion to Decimal

$$
\begin{aligned}
10010010_2 \;=\; & 1\cdot 128 + 0\cdot 64 + 0\cdot 32 + 1\cdot 16 + 0\cdot 8 \\
& + 0\cdot 4 + 1\cdot 2 + 0\cdot 1
\end{aligned}
$$

00000010	2_{10}
00010000	16_{10}
10000000	128_{10}
10010010	146_{10}

10010010	a binary number
7 6 5 4 3 2 1 0	bit numbers

The bits of a binary number are themselves numbered by the exponent of the power of 2 they multiply.
Bit number 0, multiplied by $2^0 = 1$, is the rightmost or "lowest order" bit. Bit number 7, multiplied by $2^7 = 128$, is the leftmost or "highest order" bit.

$\boxed{12.3}$ Two's Complement Numbers

Problem: How to get an 8 bit number to represent a signed integer from -128 to 127?

Answer: Use arithmetic modulo $2^8 = 256$. What is that?

Definition: $m \mod n$ is the number from 0 through $n-1$ that you subtract from m to make it an exact multiple of n (assuming $n > 0$).

If $m \geq 0$, $n > 0$, this is the remainder of m divided by n.

The number:	-128	-127	-10	-9	-1	0	1	127
Is represented by:	128	129	246	247	255	0	1	127

$\boxed{12.4}$ Two's Complement Arithmetic

$((x \bmod n) + (y \bmod n)) \bmod n = (x + y) \bmod n$
$((x \bmod n) * (y \bmod n)) \bmod n = (x * y) \bmod n$

E.g.: $(-10 \bmod 256) + (20 \bmod 256) = 246 + 20 = 266$
$266 \bmod 256 = 10 = (-10 + 20) \bmod 256$

In binary, $x \bmod 256 =$ the low order 8 bits of x.

Signed	1111 1		$\longleftarrow$carries
Binary	11110110	-10_{10}	$\longleftarrow 246_{10}$
Addition	00010100	20_{10}	
	00001010	10_{10}	

A carry into bit number 8 is discarded.

12.5 MIPS Computer Registers

32 General Registers and 3 Special Registers

`$0`	`$1`	`$2`	`$3`	`$4`	`$5`	`$6`	`$7`	`$8`	`$9`	`$10`	`$11`	`$12`	`$13`	`$14`	`$15`
`$16`	`$17`	`$18`	`$19`	`$20`	`$21`	`$22`	`$23`	`$24`	`$25`	`$26`	`$27`	`$28`	`$29`	`$30`	`$31`

`pc` `hi` `lo` (Program Counter, HI, and LO)

All these registers are 32-bits and can hold:

An unsigned binary integer from 0 through $2^{32} - 1$.

A signed binary integer from -2^{31} through $2^{31} - 1$.

Notes: $2^{10} = 1024$ so $2^{31} \approx 2{,}000{,}000{,}000$.

Register names are `$0`, `$1`, `$2`, . . . , `$31`, `pc`, `hi`, `lo`.

Register `$0` is always zero, no matter what.

Only Multiply and Divide use `hi` and `lo`.

12.6 Arithmetic Instructions

Addition Instruction (a 32 bit "word"):

31 ⋯ 26	25⋯21	20⋯16	15⋯11	10⋯ 6	5 ⋯ 0
ADD	rs	rt	rd	ADD	ADD

`rs`, `rt`, `rd` ::= `$0` | `$1` | `$2` | . . . | `$31` are registers.

Adds `rs` to `rt` and stores sum in `rd`.

Assumes registers contain signed integers.

If result is out of the range storable in a register,
"traps" in order to signal an error.

A Similar Operations: SUB: `rd` = `rs - rt`
SLT: `rd` = (`rs` < `rt`) [0 or 1]

If `rt` is `$0` then the ADD instruction copies `rs` to `rd`.

12.7 Other Arithmetic Operations

The following operation codes may be used instead of `ADD` or `SUB`:

bitwise:

```
  OR 1010      AND 1010      XOR 1010
     1100          1100          1100
  --------     ---------     ---------
     1110          1000          0110
```

The 'exclusive or' (XOR) of two bits is 1 if and only if exactly one of the bits is 1.

`ADDU`: unsigned add   `SUBU`: unsigned subtract

Unsigned add/subtract discards any carry and never traps.

12.8 Immediate Instructions

ADD Immediate:

31 ··· 26	25 ··· 21	20 ··· 16	15 ··· 0
`ADDI`	`rs`	`rt`	`immed`

Adds 16 bit signed `immed` constant value to register `rs` and stores in register `rt`.

Similar Instruction: `ADDIU`, unsigned add (never traps)
Similar Instructions but `immed` is unsigned:

`ORI`, `ANDI`, `XORI`, `LUI`

`LUI` loads `immed`$*2^{16}$ into register `rt` (ignoring `rs`).
`LUI` followed by `ORI` can load a 32-bit constant into a register.

12.9 RAM Memory

Random Access Memory (RAM) is distinct from register memory.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>. . .</td></tr>
<tr><td colspan="2">half-word</td><td colspan="2">half-word</td><td colspan="2">half-word</td><td colspan="2">half-word</td><td colspan="2">half-word</td><td></td></tr>
<tr><td colspan="4">word</td><td colspan="4">word</td><td colspan="3"></td></tr>
</table>

Bytes are 8 bits, halfwords 16 bits, words 32 bits.

Bytes have addresses `0, 1, 2, 3, 4, ...`
Halfwords have addresses `0, 2, 4, 6, 8, ...`
Words have addresses `0, 4, 8, 12, 16, ...`

12.10 Load and Store Instructions

Load Word (`LW`):

31 ··· 26	25···21	20···16	15 ··· 0
`LW`	`base`	`rt`	`offset`

`rt` = RAM word at address `base` + `offset` where `base` is a register, `offset` is a signed 16 bit constant.

Other Loads:
- `LB`/`LH` load signed 8 bit byte/16 bit halfword.
- `LBU`/`LHU` load unsigned 8 bit byte/16 bit halfword.

Stores: `SW`/`SH`/`SB` store `rt` in RAM word/halfword/byte.

Word addresses must be exact multiple of 4; halfword of 2.

$\boxed{12.11}$ The Program Counter

Example: Add 1 to word at address 100 (denoted by `W(100)`).

`pc`

LW	\$0	\$24	100	\$24 = W(100)
ADDI	\$24	\$24	1	\$24 = \$24+1
SW	\$0	\$24	100	W(100) = \$24

Processor Action (repeated endlessly):

(1) Fetch word addressed by `pc`.

(2) Execute fetched word as instruction.

(3) If (2) did <u>not</u> change `pc`, then set `pc` = `pc` + 4.

12.12 Branches

Example: `$2 = max ( $4, $5 )`

SLT	$4	$5	$24	SLT	SLT	$24 = ($4<$5)
BGTZ	$24	BGTZ	3 (offset)			if ($24>0) goto .+12
ADD	$4	$0	$2	ADD	ADD	$2 = $4
BLEZ	$0	BLEZ	2 (offset)			goto .+8
ADD	$5	$0	$2	ADD	ADD	$2 = $5

Branches change the pc if a test of a register is true.

BGTZ: if (reg $>$ 0) pc $=$. $+$ 4*offset

BLEZ: if (reg $\leq$ 0) pc $=$. $+$ 4*offset

'.' denotes location of branch instruction.

Offset is signed 16 bit number and is multiplied by 4.