# Number System

* Sign-and-Magnitude

1-bit sign: 0 for +, 1 for –; n-1 bit magnitude.

Largest Value: ; Smallest Value: ; Zeros:

* r-1’s complement

For positive number, complement is itself.

For negative, n-bit integer, m-bit fraction:

构造一个全是的数(eg. 111…111)，然后减掉对应的正数，得到负数补码。

对二进制是各位取反。

* r’s complement

For positive number, complement is itself.

For negative, n-bit integer, m-bit fraction:

构造一个全是0的数，然后减掉对应的正数，得到补码。

Smallest: (1000…000)，only one 0

If there is a carry out of MSB, ***add 1*** to result.

* Excess Notion

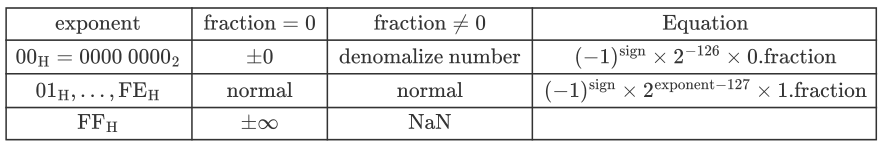
Excess-7: 0000 map to -7, 平移过去

* Floating Point

sign | exponent | mantissa

single-precision(32-bit): 1-8-23, excess-127, hidden-1

double-precision(64-bit): 1-11-52, excess-1023, hidden-1



* Tips:

1. round off(四舍五入，比如0b0.001得到0b.01)

2. Overflow: pos + pos = neg; neg + neg = pos;

# MIPS

R-Format: opcode(6,all 0)+ rs(5), rt(5), rd(5), shamt(5), funct(6)

add, sub, ***slt***, ***sll***(sll rd, rt, shamt)***.***

I-Format: opcode(6) + rs(5) + rt(5) + immedate(16)(Signed Number)

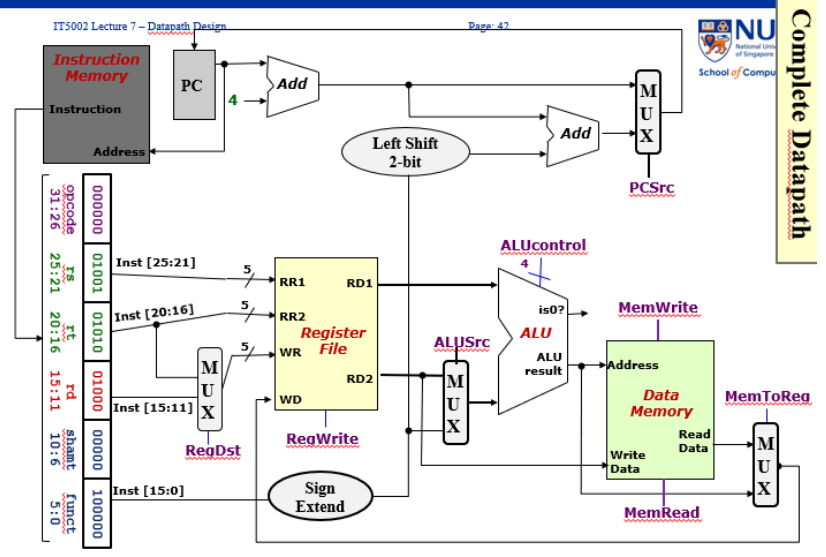
Branch(bne, beq)

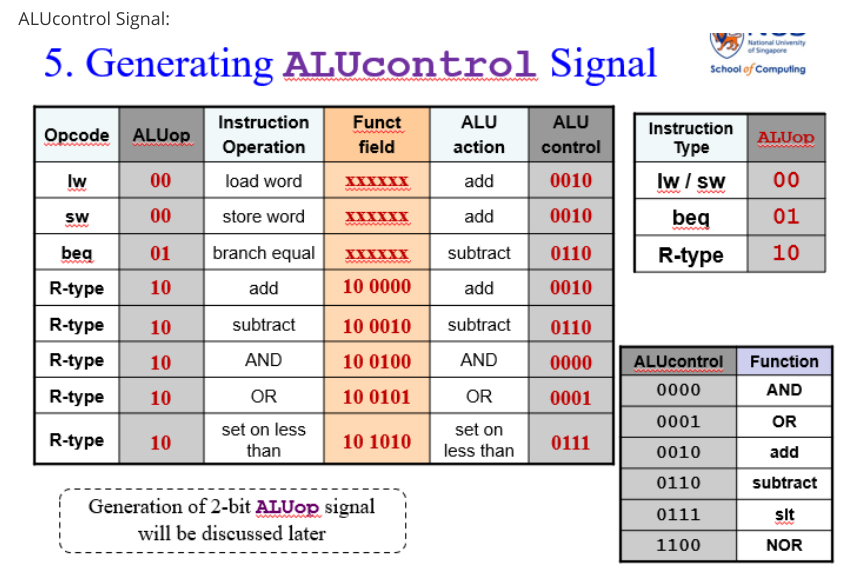
Branch range: words or bytes.

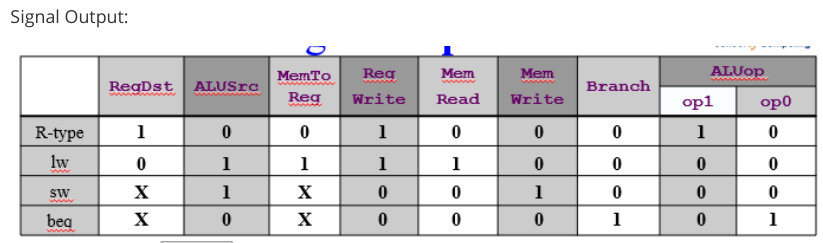
J-Format: opcode(6) + target add(26)(Unsigned)

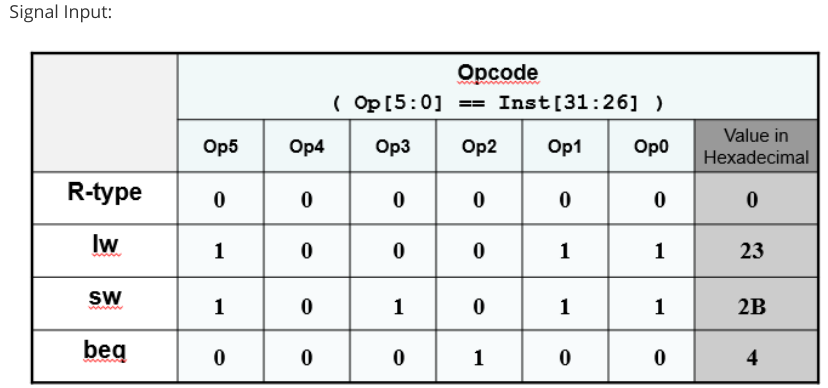
Address

Jump range: 256MB, bytes









# Pipeline

| Instruction | IF | ID | ALU | MEM | WB |
| --- | --- | --- | --- | --- | --- |
| Arithmetic | X | X | X |  | X |
| Branch | X | X | X |  |  |
| Load | X | X | X | X | X |
| Store | X | X | X | X |  |

* Single cycle:

Cycle time:

Time : 每条指令花费时间的和

* Multi cycle:

1. 计算cycle time: 每个stage中时间最长的为cycle time.

2. 计算每条指令的时间，根据cycle time \* (#stage)

3. 计算CPI，平均时间

4. Total time: #Instructions \* CPI

* Pipeline:

1. 计算cycle time: (max(每个stage时间) + delay time)

2. Total time: (I+N-1) \* cycle time; (N-1)是fill time

Speedup: single cycle or Multi-cycle花费时间/ pipetime 时间

引入额外的寄存器:

IF/ID:

* PC+4(Branch), rs, rt, rd, shamt, func, imm(16)

ID/EX:

* PC+4, RD1, RD2, rt, rd(存疑，我觉得此处可以决定DstReg了), Imm(32)
* MtoR, RegWr, MemR, MemW, Branch, RegDst, ALUsrc, ALUop

EX/MEM:

* BrcTgt(PC+4+4\*Imm), isZero, ALUres, RD2, DstReg(rt or rd)
* MtoR, RegWr, MemR, MemW, Branch

MEM/WB:

* MemRes, ALUres, DstReg
* MtoR, RegWr

# Cache

**Hit & Miss:**

Hit time: Time to access cache.

Miss penalty: Hit time + time to memory

**Types of Miss**:

* Cold/Compulsory Miss: on the first time access to a block. (如果cache block 从来没有装载过)。First reference miss, cold start miss。
* Conflit Miss: 曾经装载过，但是换出了。Collision miss, interference miss. Fully Associative Cache不会发生。
* Capacity Miss: 全部cache容量满了。

**Policy:**

* Write Policy: Cache hit
  + Write-Through: 同时写入Cache 和MEM, using write buffer.
  + Write-Back: add a dirty bit, only write to memory when kicked out.
* Write Miss Policy: Cache miss
  + Write-Allocate: Load block to cache. 变成了cache hit情况.
  + Write-Around: Directly write to MEM.
* Replacement:
  + Least Recently Used(LRU): 需要记录最近什么时候用过
  + FIFO:先进先出
  + Random Replacement
  + Least Frequently Used:使用频率最低

**N-Way Set Associative Cache**: N-way: 一个set里有n路(可以放n个不同的tag)

1. Block Size: bytes, N-bit

2. Number of **Cache Set**: cache 总容量/(n\*block-size) = , M-bit

3. Tag: 32 – (N+M) bits

Fully Associative Cache: N=Block Number, a block can be stored in anywhere.

No Set Index

(Directly mapped: 一列; N-way: n 列, set index行; Fully: N 列，1行)