2019 02 04

Ex (not MIPS)

x = a + b - c;

```
"add" t, a, b   #t <- a+b
"sub" x, t, c   #t <- (a+b) - c
```

1st item is always destination

See figure 1 in slack

```
read MEM[a]             # 8 cycles
read MEM[b]             # 8 cycles
add above 2             # 1
store result at MEM[t]  # 8
read MEM[t]             # 8
read MEM[c]             # 8
sub ...                 # 1
store result at MEM[x]  # 8
//TOTAL of 50 Cycles
```

FSB Front Side Bus.

CPU operates at 3.2GHz, FSB operates at 400MHz, meaning one read of memory takes 8

Ways to improve this problem:

1. increase the speed of the FSB
   - the problem with this is that finite distance scale of the FSB limits speed improvements
2. Package things shorter
   - some advantages but FSB also has to attach to other items
3. create a cache of processor local RAM
   - not covered in this course.
   - programmer transparent, does not know whether something is in cache or not. CPU decides what is copied into the cache.
   - registers: these are a much smaller than the cache, for high priority variables. Programmer decided/controlled.

---

https://en.wikipedia.org/wiki/Northbridge_(computing)

FSB bottleneck is obvious in diagram. ~Note the screen resolution bottleneck example is strait from the textbook~

Behold the MIPS

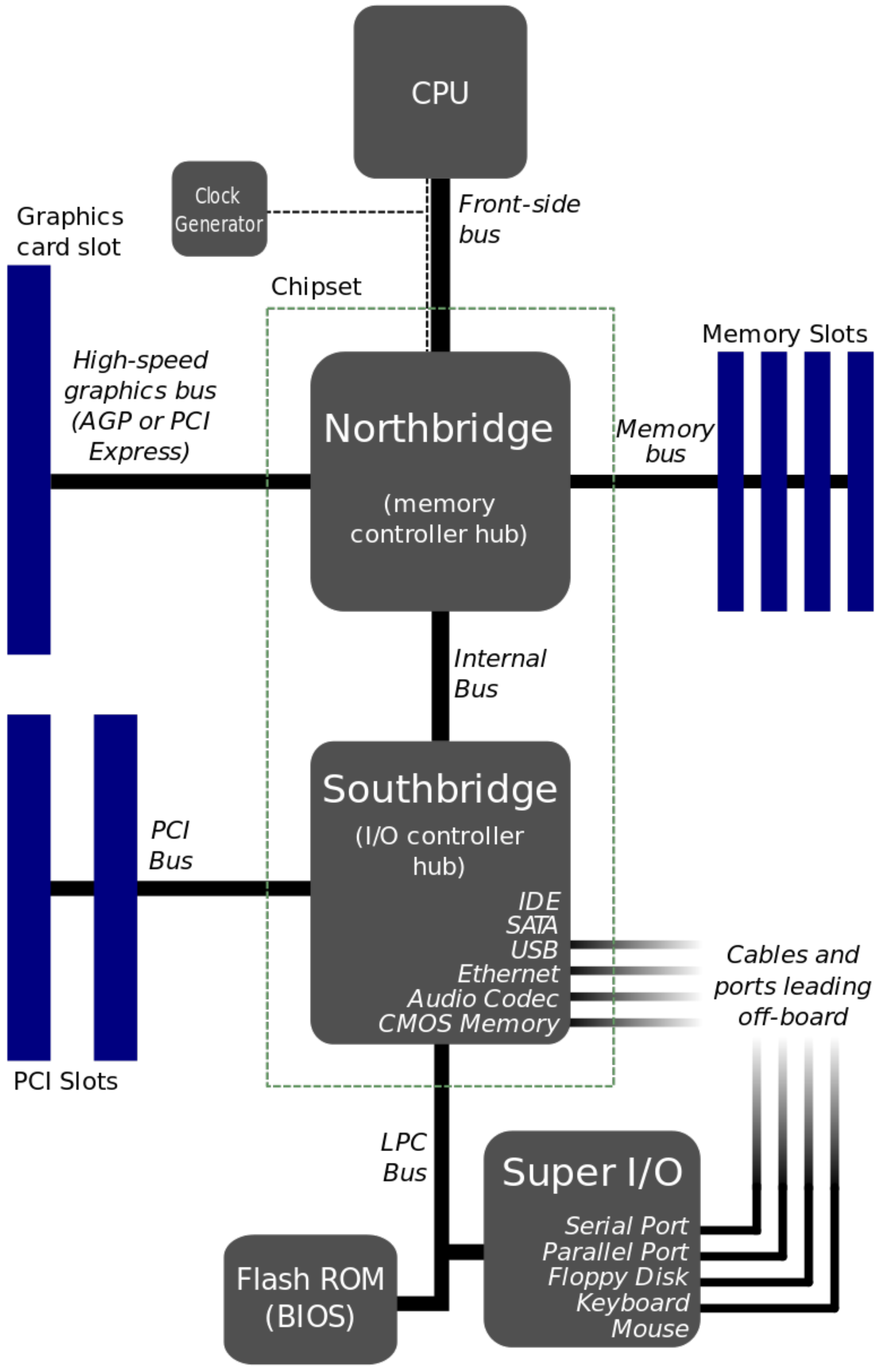

Figure 1: northbridge southbridge diagram

Conventions: Registers on MIPS (32 registers of 32 bits each)

- general purpose registers
- there are conventions for what these 'general' registers

Register name	value stored in register
`$s0, $s1, ..., $s7`	HLL variables
`$t0, $t1, ..., $t9`	Temp. variables
`$zero`	Always stores 'zero'
(thirteen more. . .)	discussed later

Naming Conventions	Description
`rd, rs, tt`	registers (any of 32 gen purpose)
`imm`	immediate, ie, 2's complement constants (16-bit)

Instructions (MIPS)

```
add   rd, rs, rt      #rd <- rs + rt
sub   rd, rs, tt      #rd <- rs - rt
addi  rt, rs, imm     #rt <- rs _ imm
```

Suppose you are going to write a MIPS program to execute the following C code

```
x=a + b - c + 5;
```

```
#allocate: a-> $30  b ->$s1 c->$s2  x->$s3
add   $t0, $s0, $s1   #$t0 <- a+b
sub   $st1, $t0, $s2  #$t1 <- a+b - c
addi  $s3, $t1, 5     #x <- a+b - c + 5 NOTE: USING 5 literally
```

RAM: Random Access Memory

See figure 2 slack

Memory instructions:

```
lw <destination>,<source>   #load word
sw <source>,<destination>   #store word
```

destination is a register (always) source (imm (register) i.e the address you are storing is a immutable)

Example implement the following C code in MIPS

```
x #<-arr[i] + y;
```

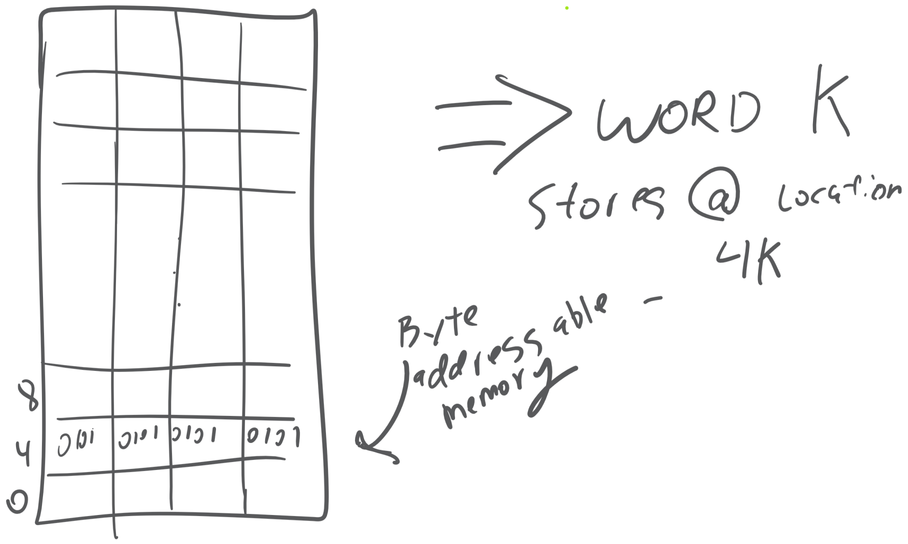

Figure 2: ram illustration

```
#Alloc: i -> $s0, y -> $s1 x -> $s2
#Alloc: arr -> $s3 # base address of arr`
add   $t1, $s0, $s0   # t1 <-2i
add   $t1, $t0, $t0   # t1 <-4i
add   $t1, $t1, $s3   # t1 <- addr. of arr[i]
lw    $t0, 0($t1)     # t0 <- arr[i]
add   $  , $t0, $s1   # x  <- a
```