

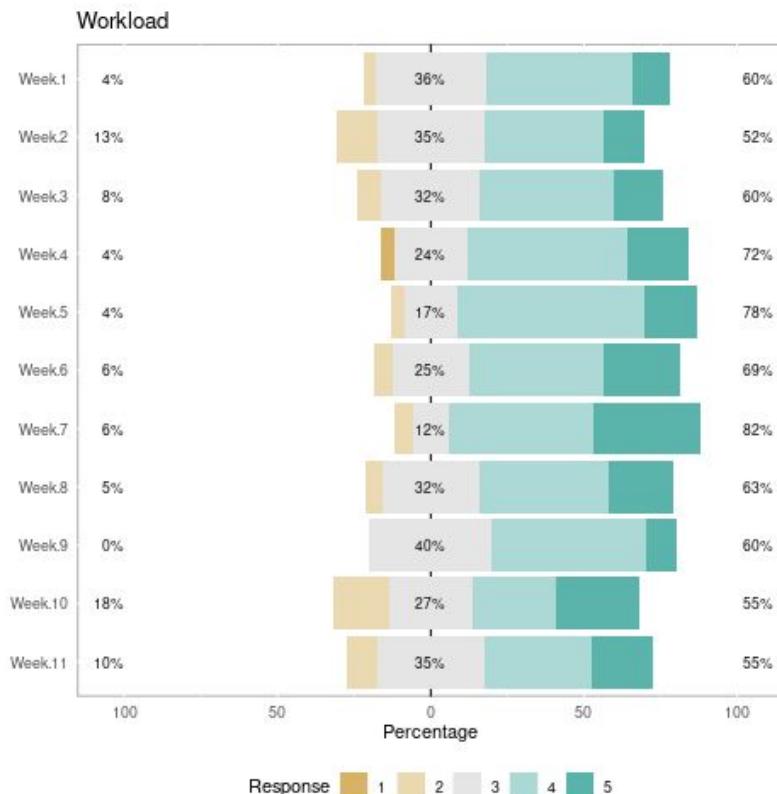
# 0x15 - Other Architectures

ENGR 3410: Computer Architecture

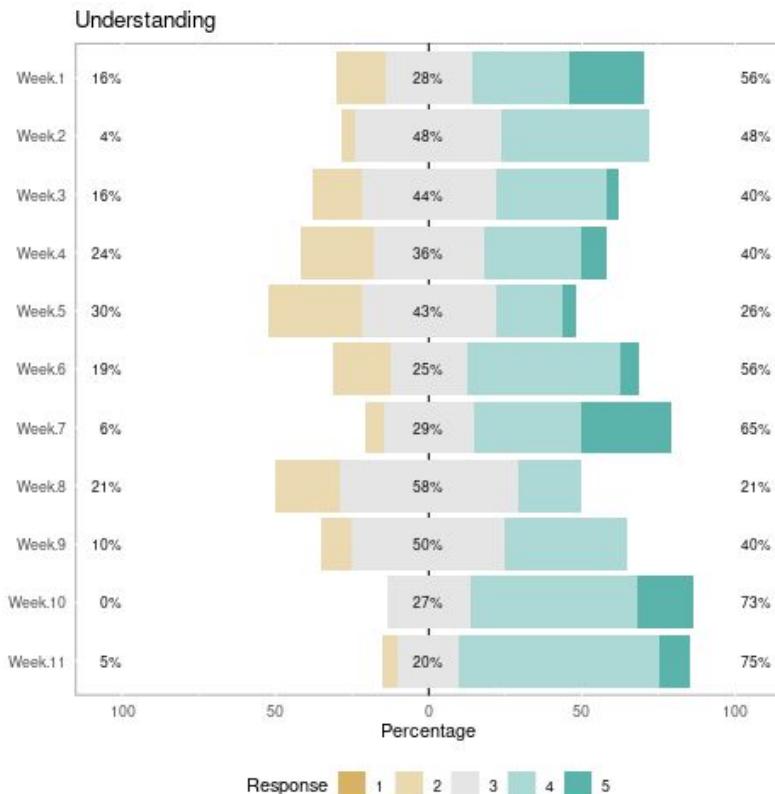
Jon Tse

Fall 2020

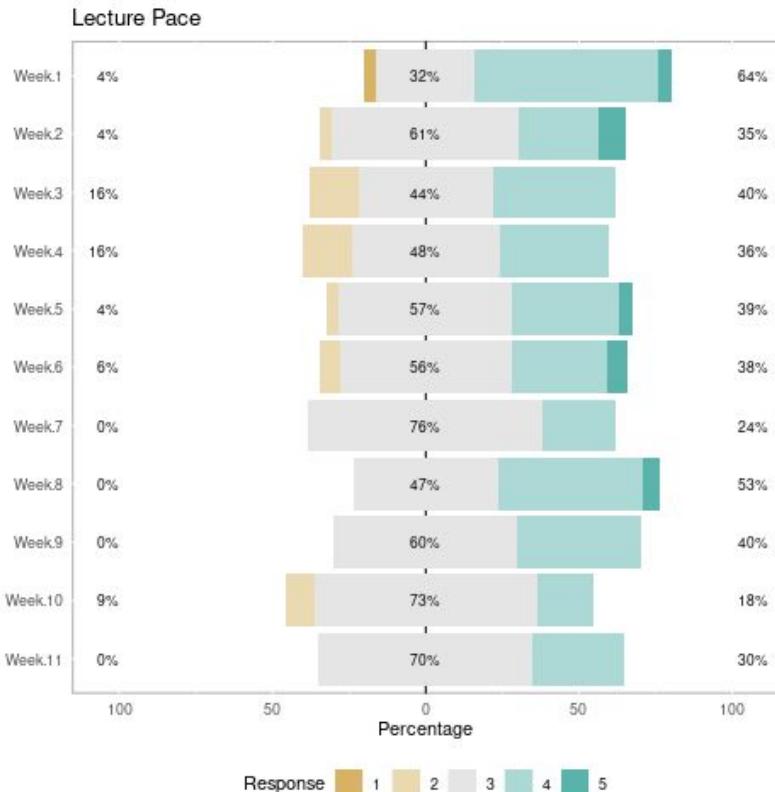
# Feedback - Workload



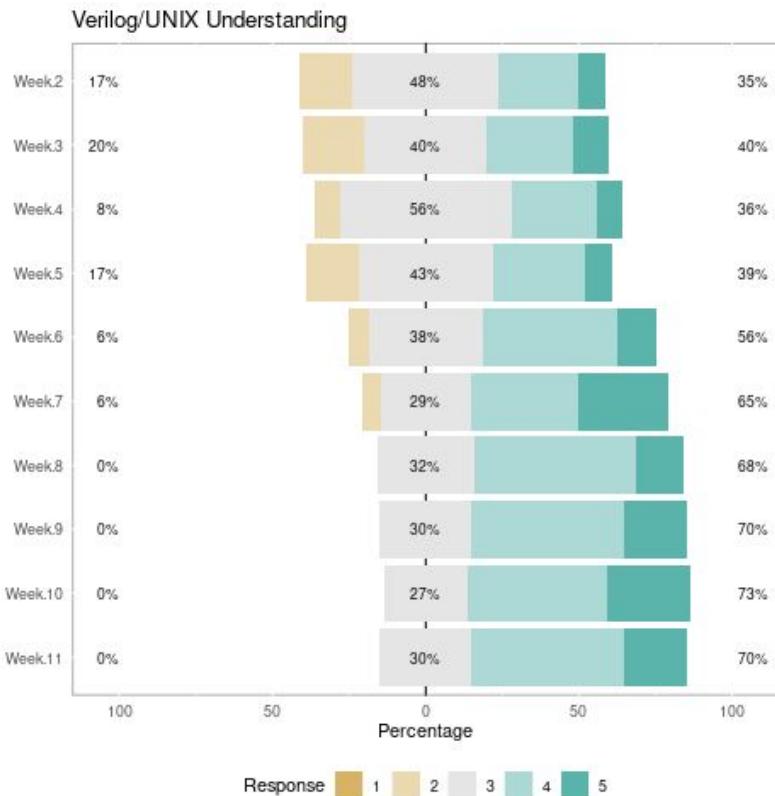
# Feedback - Understanding



# Feedback - Pace



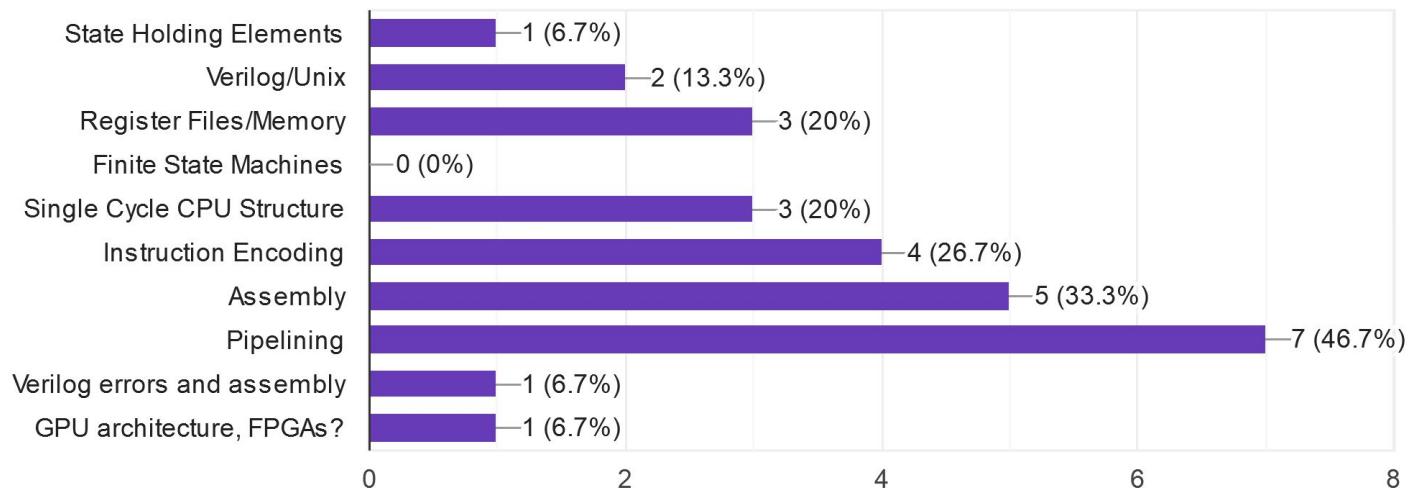
# Feedback - Tools



# Feedback - More Time...

I wish we spent more time on...

15 responses



# Feedback - Top of Mind

Anything else on your mind?

3 responses

AAAAAAHHHHH I am so overwhelmed with work. It is not so much this class just everything is hitting at once and does not appear to be slowing down. I stayed up until 6am programming last night.

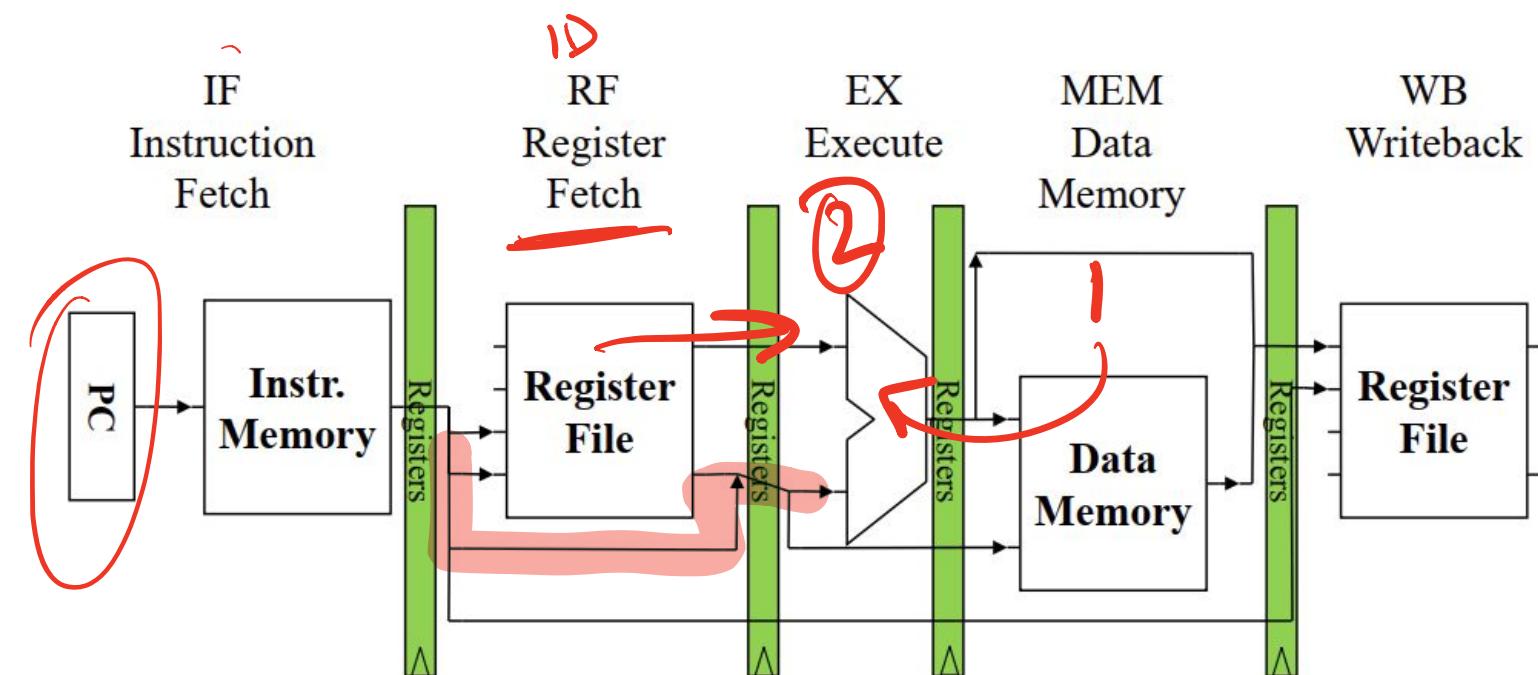
Thanks Jon.

This is a fun class!

# Housekeeping

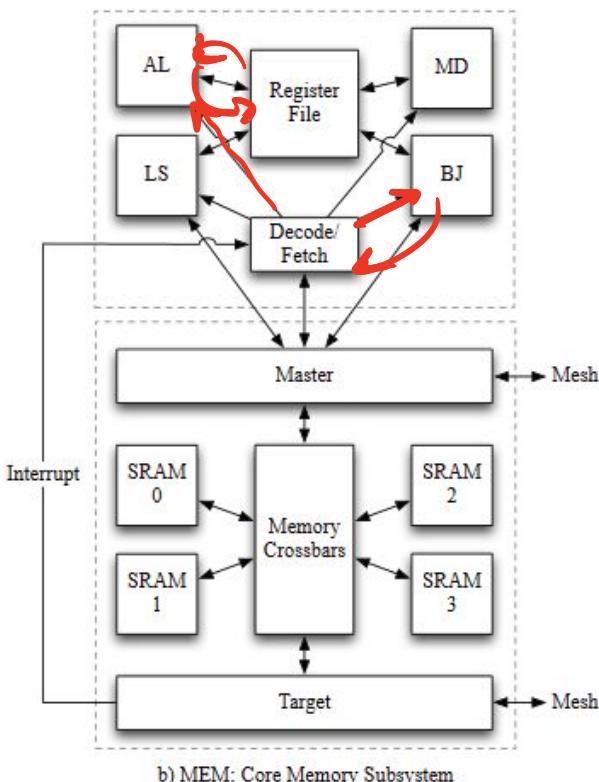
- Sign up for a Final Project Time Slot!
- Keep these slots for exam period?
- Reminder to petition NINJAs for advising
- Ping us early and often!

# Our CPU Architecture

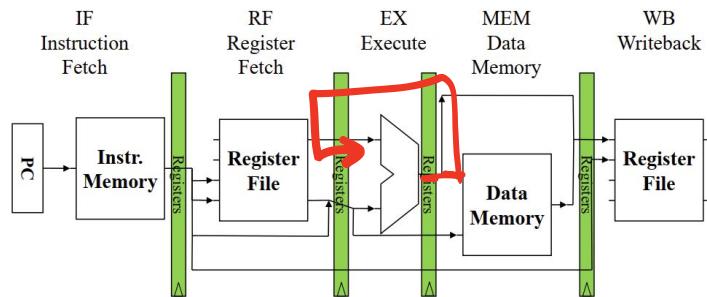


# NanoMesh - Kilo Core System

a) CPU: Core Compute Subsystem



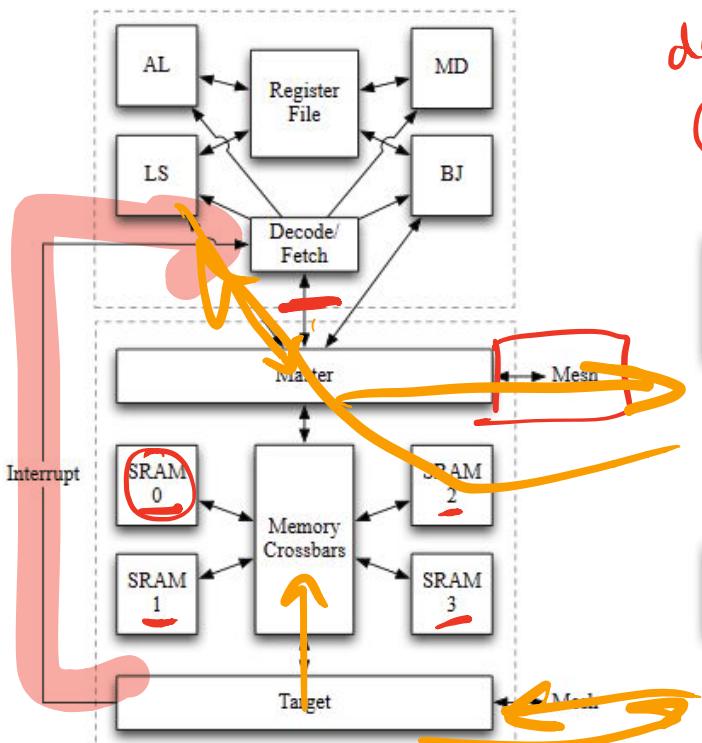
b) MEM: Core Memory Subsystem



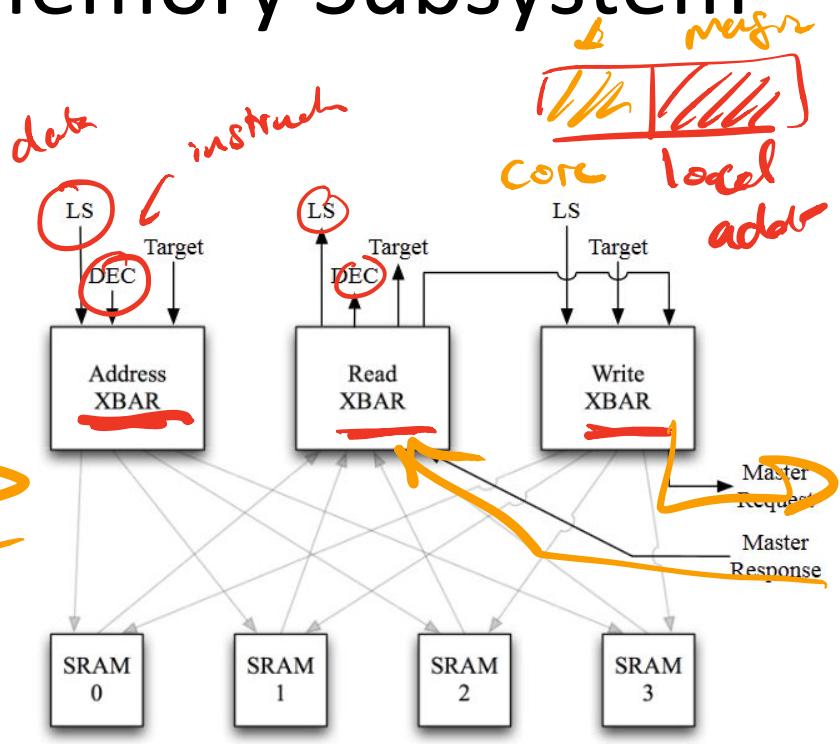
Unit	R3000	New
DEC	NOP RFE SYSCALL	HALT WAIT
AL	ADDU ADDIU SUBU ANDI ORI OR NOR XOR XORI LUI SLL SLLV SRA SRL SRLV SLT SLTI SLTU SLTIU	
MD	MULT MULTU DIV DIVU MFHI MFLO MTHI MTLO	
BJ	BEQ BGEZ BGTZ BLEZ BLTZ BNE BGEZAL BLTZAL J JAL JALR JR MFCO MTCO	
LS	LB LBU LH LHU LW SB SH SW	CF4 CF16 CT4 CT16
Omitted	ADD ADDI SUB LWL LWR SWL SWR	

# NanoMesh - Memory Subsystem

a) CPU: Core Compute Subsystem

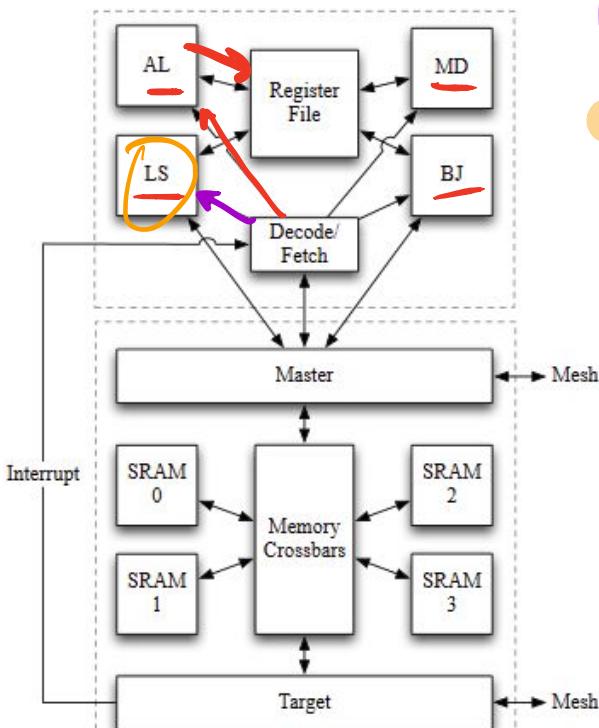


b) MEM: Core Memory Subsystem



# NanoMesh - Register Bypass

a) CPU: Core Compute Subsystem

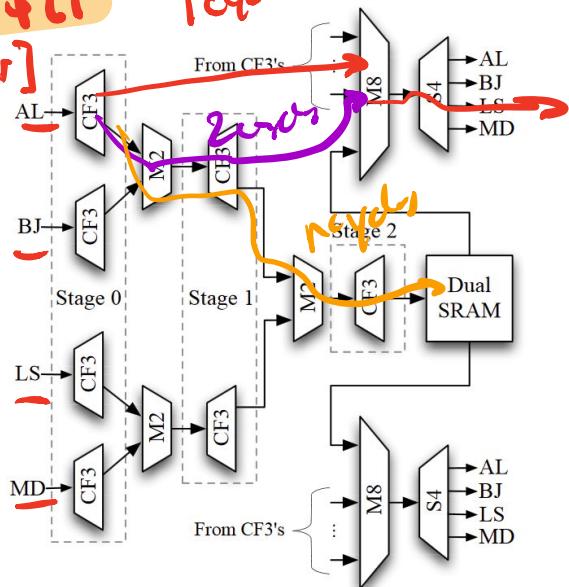


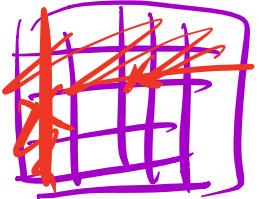
b) MEM: Core Memory Subsystem

*add \$t1  
lw \$t0, 4(\$t1)*

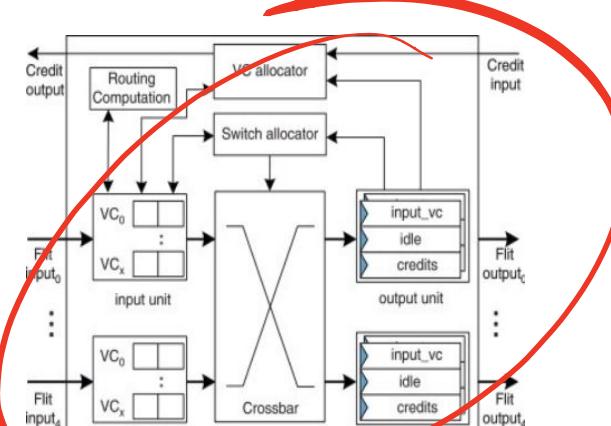
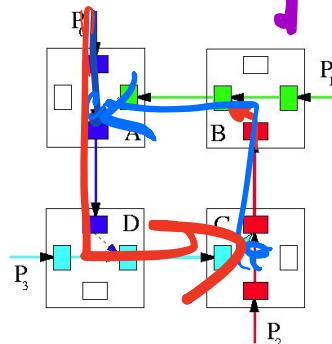
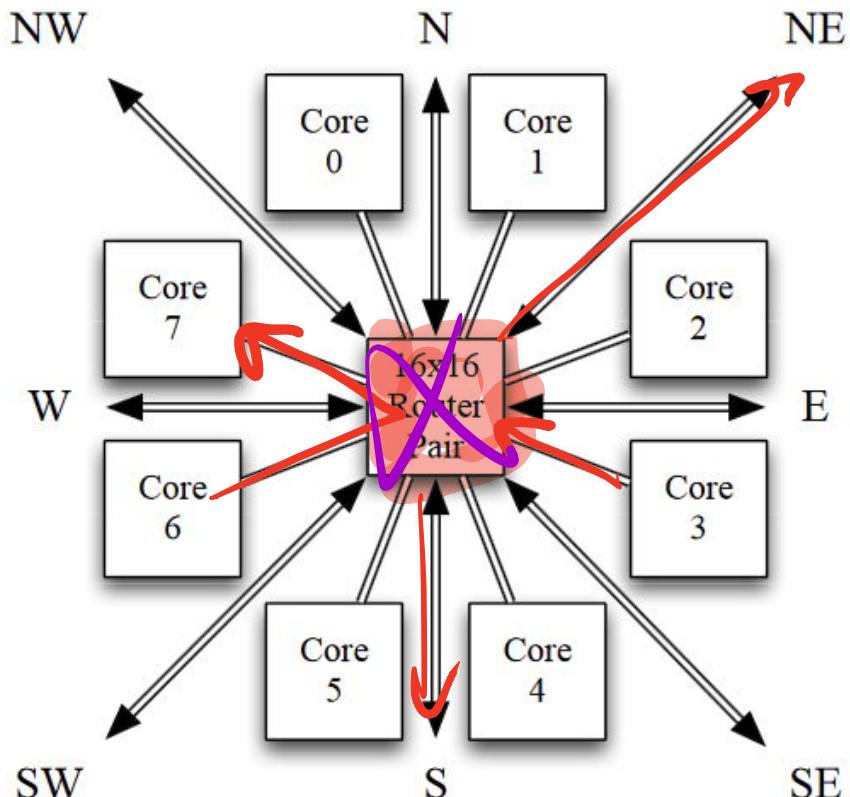
*addr = 4 + \$t1  
t0 = mem[addr]*

*1 cycle*

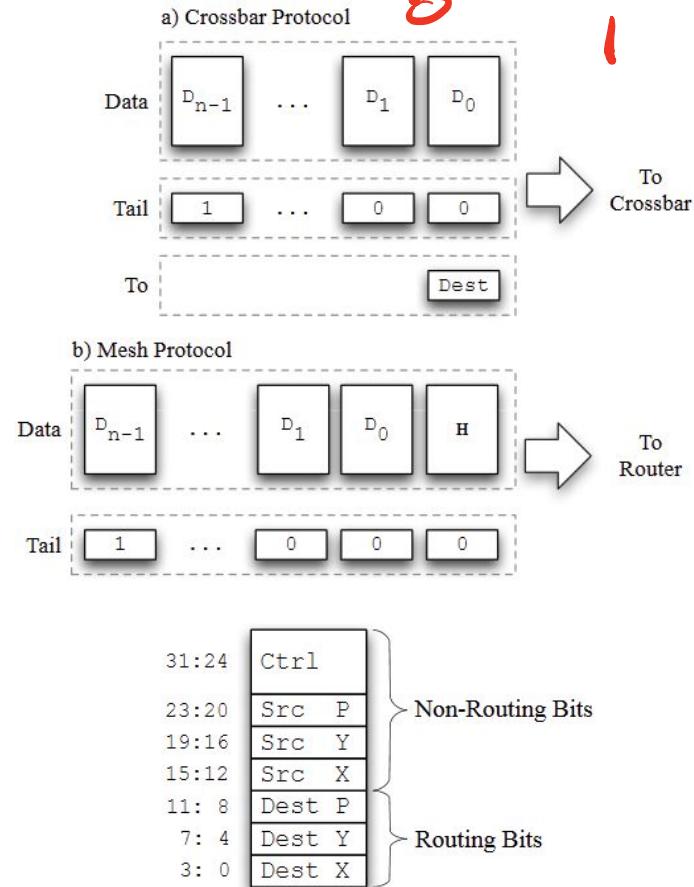
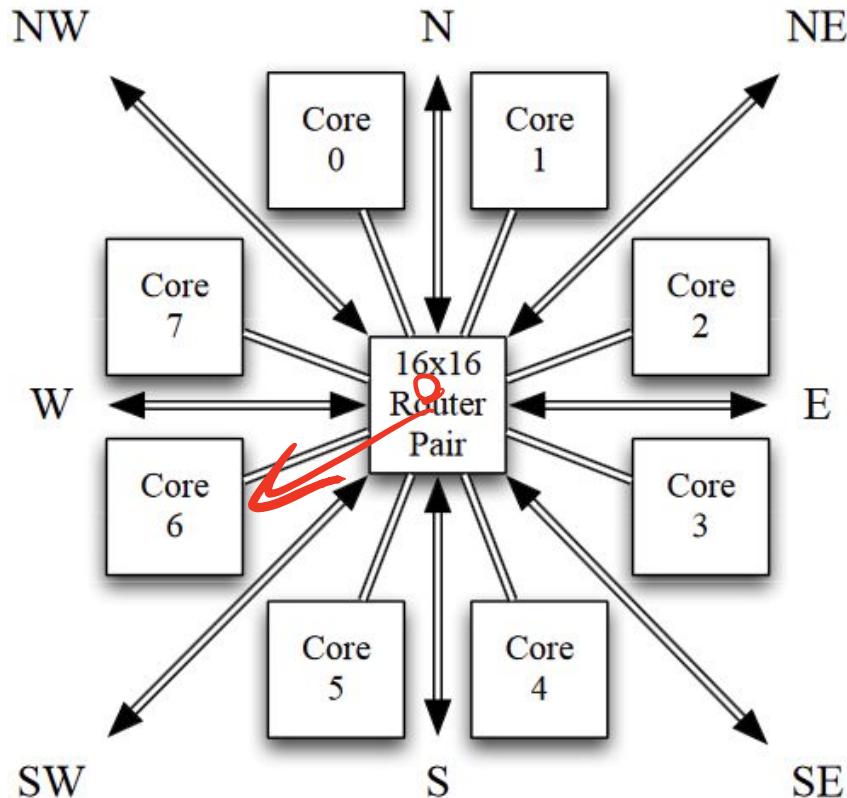




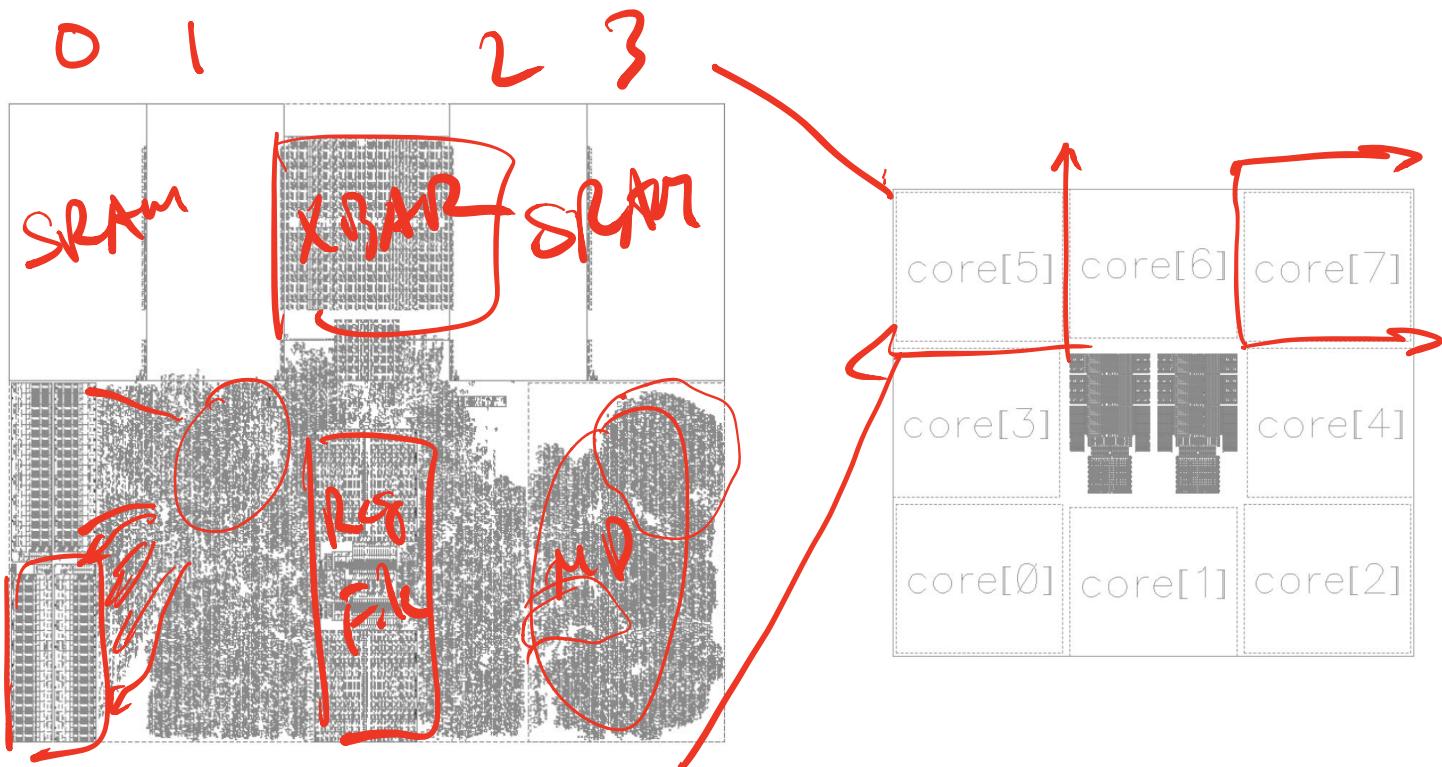
# NanoMesh - Mesh



# NanoMesh - Mesh



# NanoMesh - Floorplan



# NanoMIPS - New Instructions

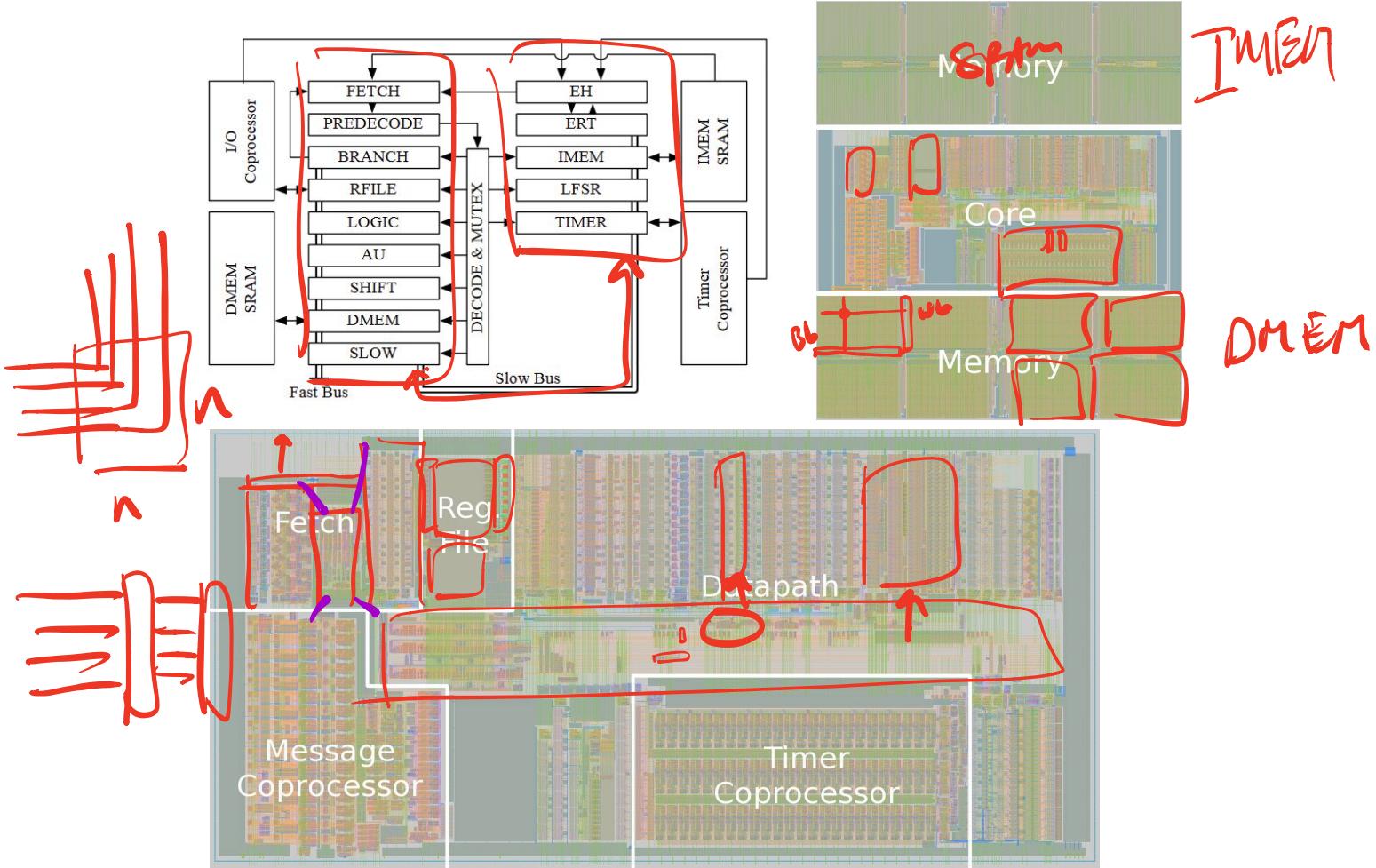
- ▶ WAIT — Stalls fetch until dirty memory flag set
  - ▶ Prevents busy-waiting
  - ▶ Allows for rudimentary message-passing constructs
- ▶ HALT — Stalls instruction fetch until Interrupt
  - ▶ NanoMIPS cores bootstrap into HALT state
  - ▶ Exposes clock-gating like behavior directly to the programmer
- ▶ Memcopy Instructions
  - ▶ CF4, CF16 — Copies 4 or 16 words from local memory to remote memory address.
  - ▶ CT4, CT16 — Copies 4 or 16 words from remote memory to local memory.
  - ▶ Addresses must be block-size aligned.
  - ▶ Supports all addressing modes
  - ▶ Supports dirty memory flag setting

# NanoMesh - Example Program

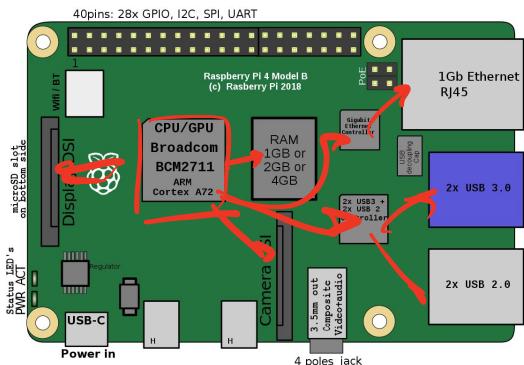
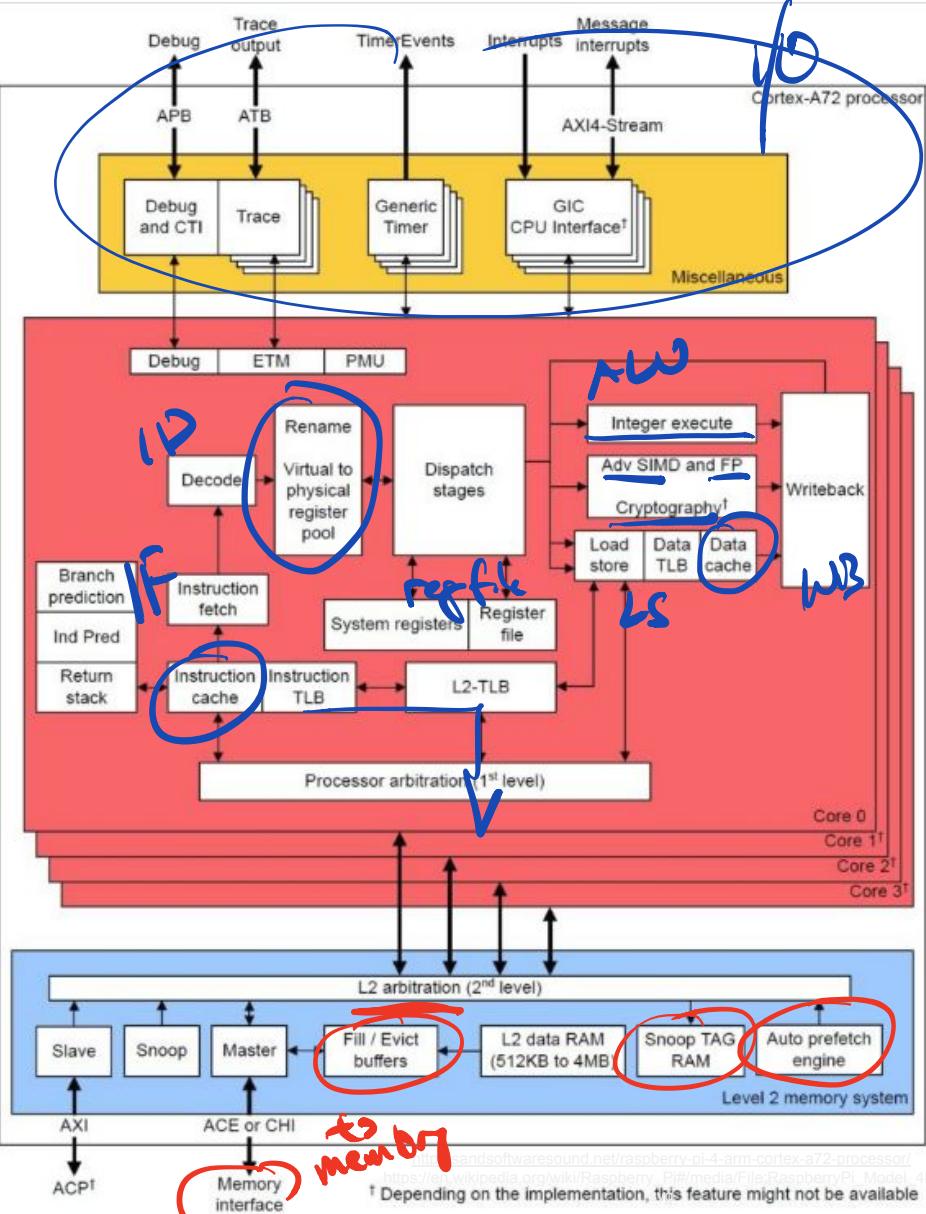
- | Core A — Control            | Core B — Worker          |
|-----------------------------|--------------------------|
| 1. Write Program to Core B  | 1. HALT                  |
| 2. Fire Interrupt on Core B | 2. Initialize            |
| 3. Send data to Core B      | 3. WAIT                  |
| 4. WAIT                     | 4. Do work on data       |
| 5. Continue execution       | 5. Send result to Core A |
|                             | 6. HALT                  |
- 
- The diagram illustrates the interaction between Core A (Control) and Core B (Worker). Core A initiates the process by writing a program to Core B, firing an interrupt, sending data, and then waiting. Core B responds by initializing, waiting, performing work on the data, sending the result back to Core A, and finally halting. Hand-drawn red sketches of people are at the bottom.



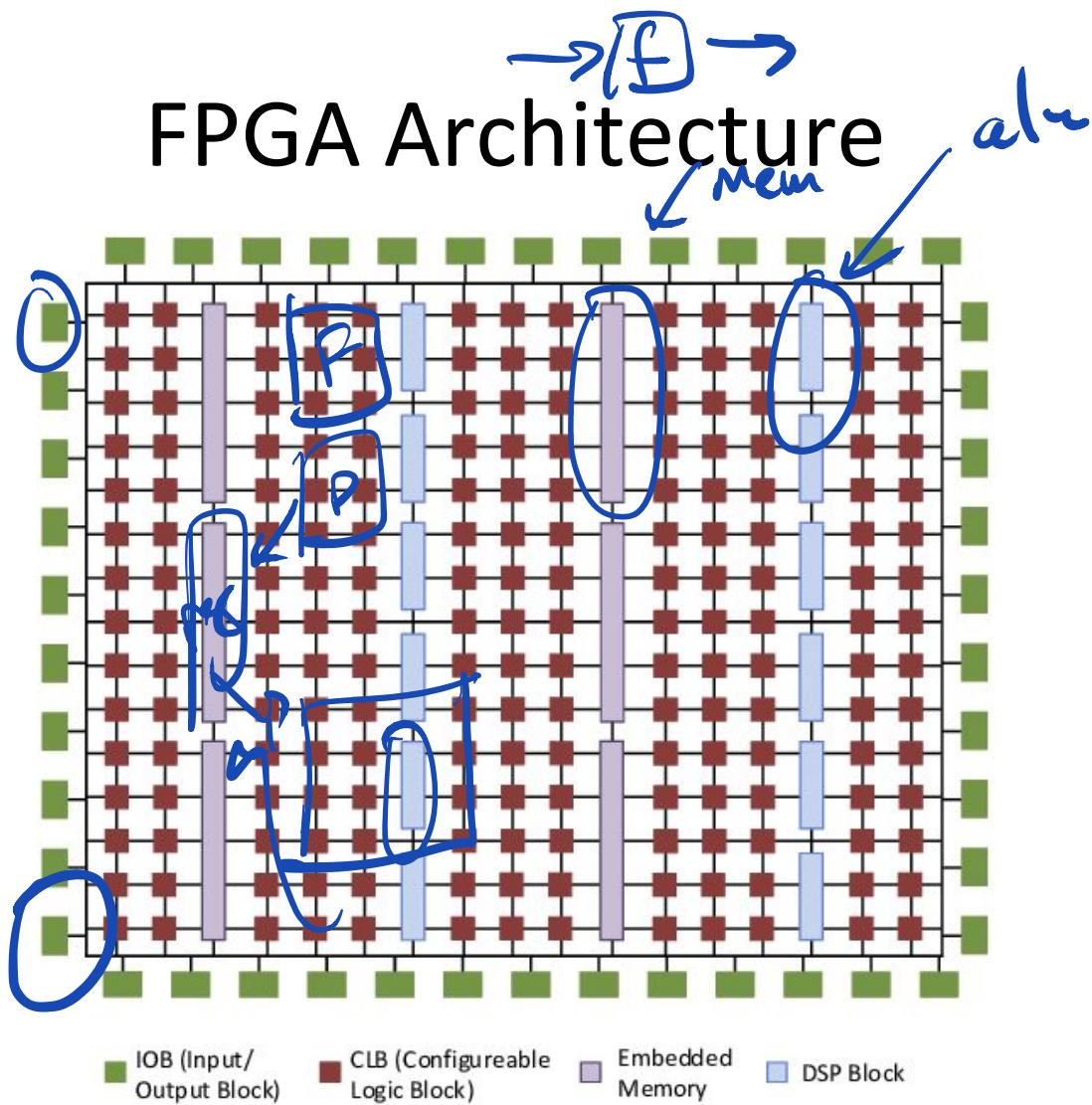
# ULSNAP - Low Power CPU



# ARM Cortex A72



# FPGA Architecture



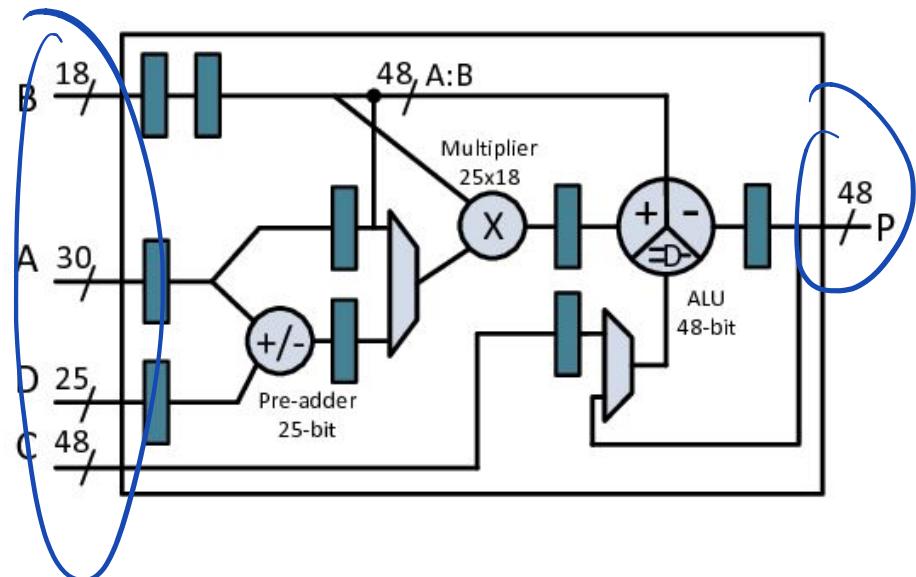
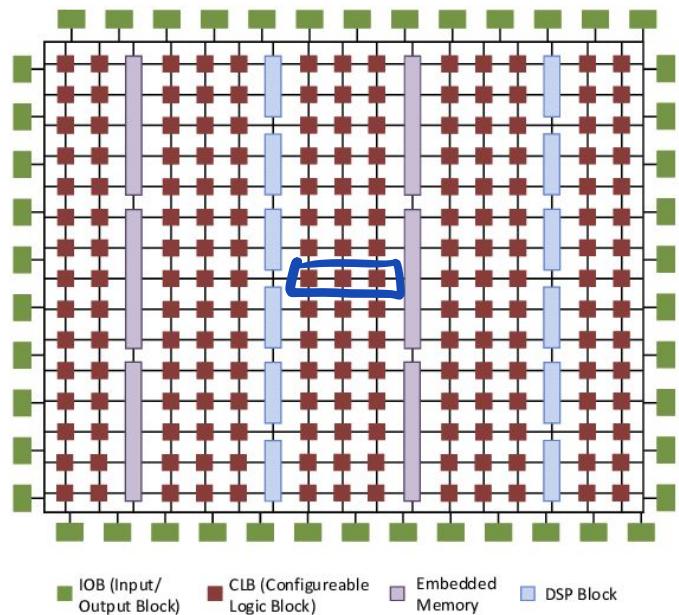
■ IOB (Input/  
Output Block)

■ CLB (Configureable  
Logic Block)

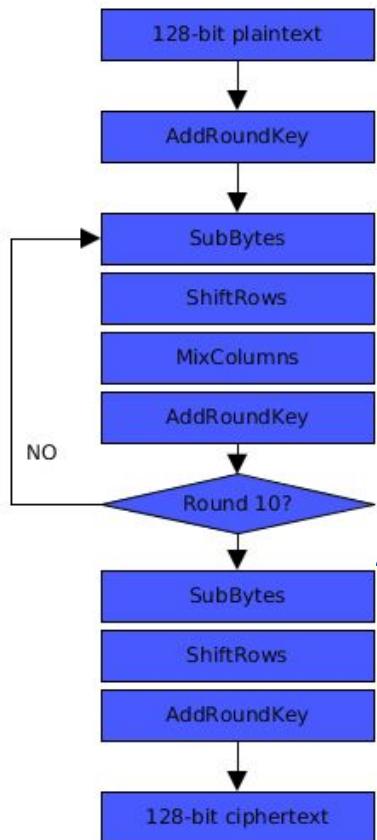
■ Embedded  
Memory

■ DSP Block

# FPGA DSP Block

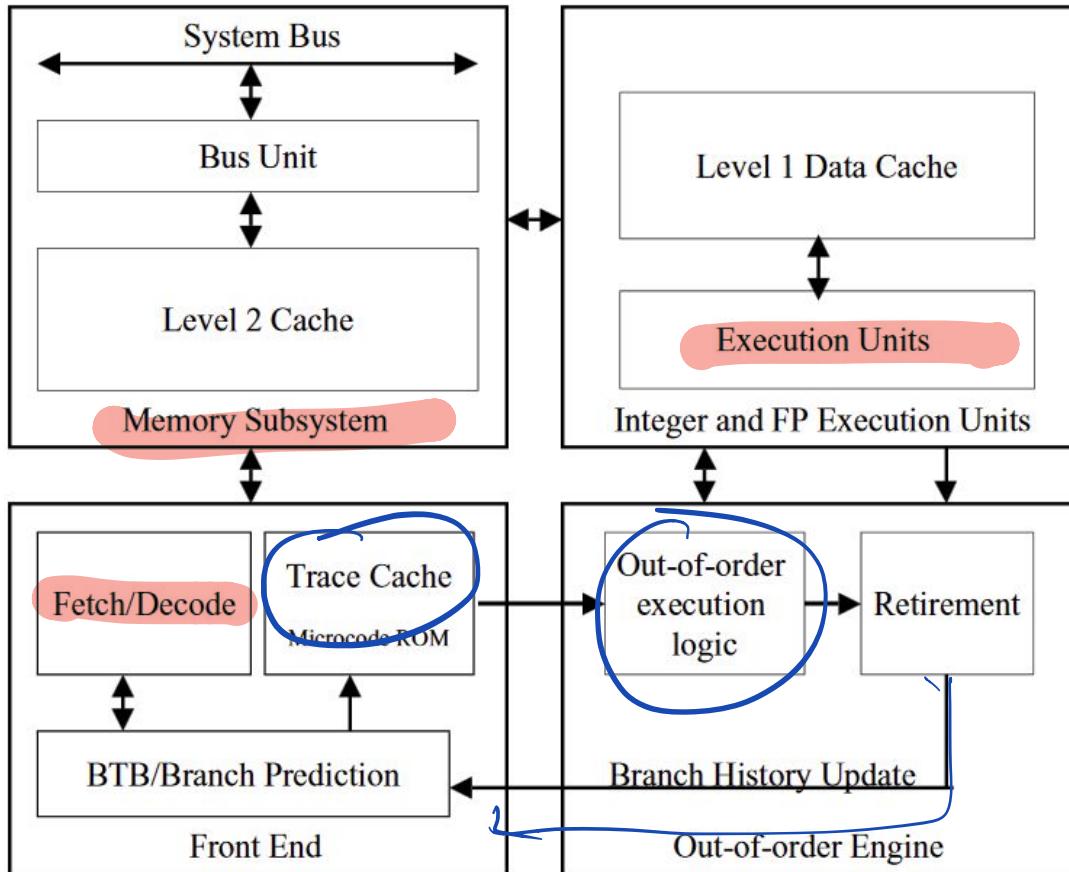


# AES on FPGA



```
; **** ShiftRows and SubBytes ****
; **** F(16)((B*x)[(i%4)+(((i/4)-(i%4))%4)*4]=S(((B*s)[i]));
pushad
    mov    cl, 16
shift_rows:
lodsb                      ; al = S(s[i])
call   sub_byte
push   edx
mov    ebx, edx             ; ebx = i%4
and    ebx, 3
shr    edx, 2                ; (i/4 - ebx) % 4
sub    edx, ebx
and    edx, 3
lea    ebx, [ebx+edx*4]      ; ebx = (ebx+edx*4)
mov    [edi+ebx], al          ; x[ebx] = al
pop    edx
inc    edx
loop   shift_rows
popad
```

# Pentium 4



# Pentium 4

## Basic Pentium III Processor Misprediction Pipeline

1 Fetch	2 Fetch	3 Decode	4 Decode	5 Decode	6 Rename	7 ROB Rd	8 Rdy/Sch	9 Dispatch	10 Exec
------------	------------	-------------	-------------	-------------	-------------	-------------	--------------	---------------	------------

# Pentium 4

## Basic Pentium III Processor Misprediction Pipeline

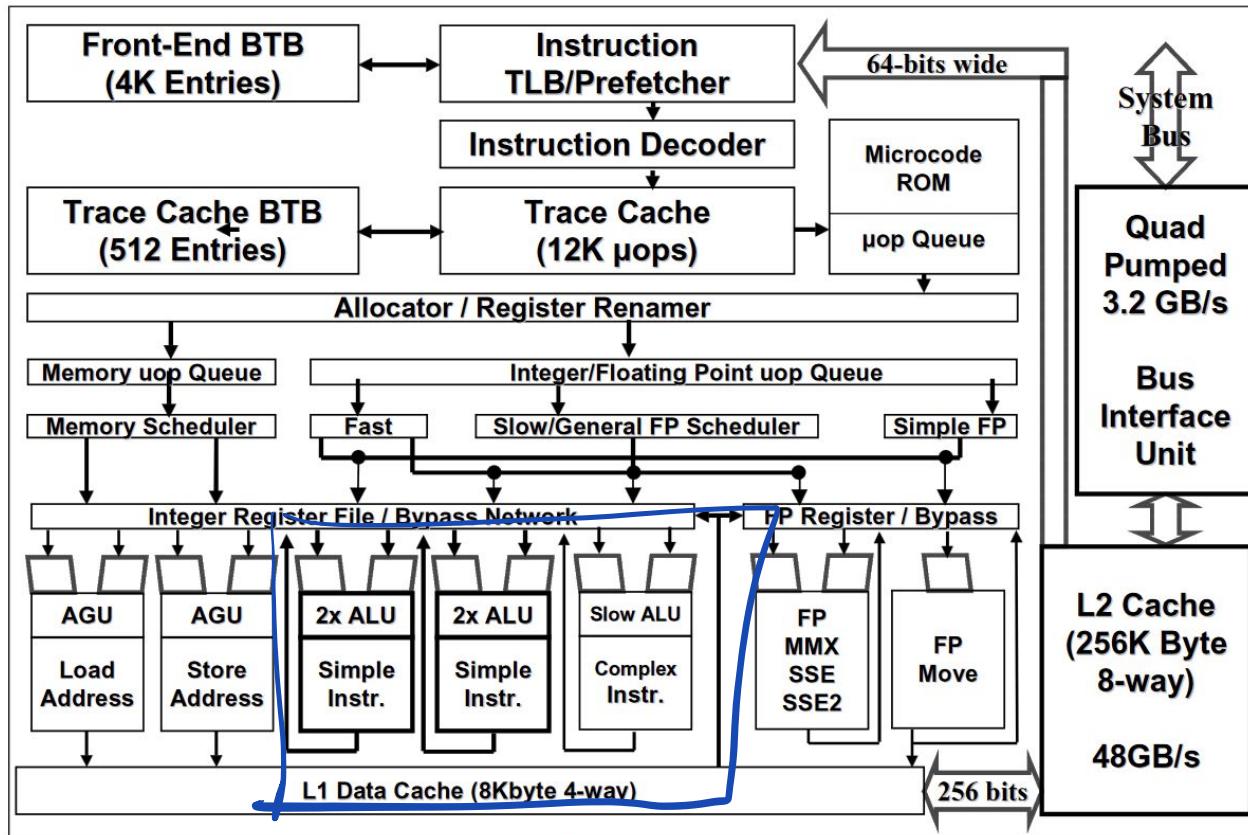
1 Fetch	2 Fetch	3 Decode	4 Decode	5 Decode	6 Rename	7 ROB Rd	8 Rdy/Sch	9 Dispatch	10 Exec
------------	------------	-------------	-------------	-------------	-------------	-------------	--------------	---------------	------------

## Basic Pentium 4 Processor Misprediction Pipeline

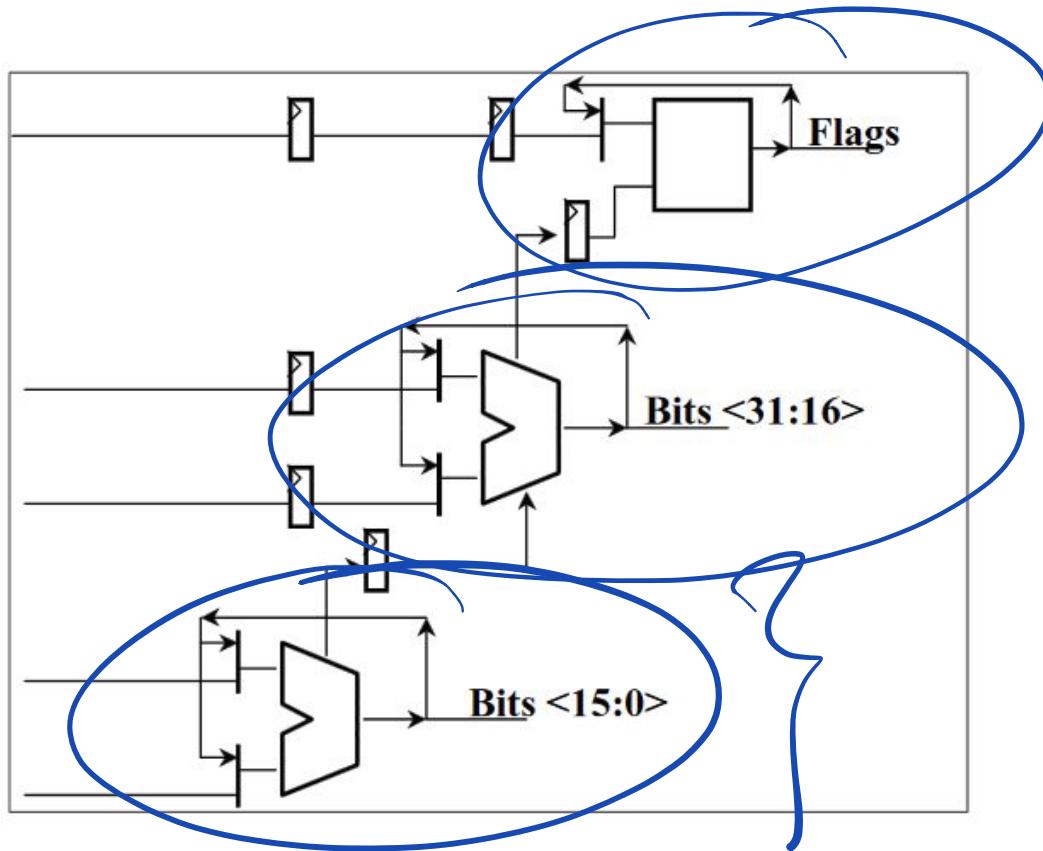
1 TC Nxt IP	2 TC Fetch	3 Drive	4 Alloc	5	6	7 Rename	8	9 Que	10 Sch	11 Sch	12 Sch	13 Disp	14 Disp	15 RF	16 RF	17 Ex	18 Flgs	19 Br Ck	20 Drive
----------------	---------------	------------	------------	---	---	-------------	---	----------	-----------	-----------	-----------	------------	------------	----------	----------	----------	------------	-------------	-------------



# Pentium 4



# Pentium 4



# GPUs *3d fx*

