PDF notes

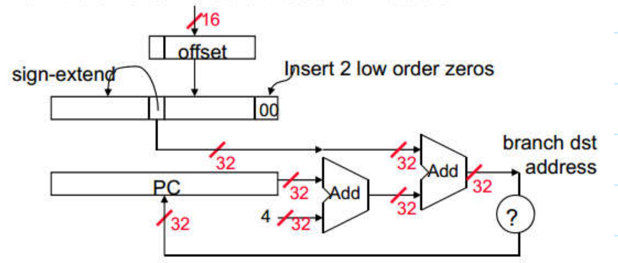
Monday, October 5, 2015 10:14 AM

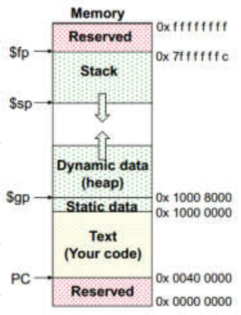
Chapter 1 pdf

- Define the dif between Servers, desktop, and embedded computers
- List the 5 components of a computer
  - Input, output, memory, datapath, control
- What is ISA and why is it used
  - ISA stands for instruction set architecture. It provides an abstraction between the hardware and the lowest level software. It allows varying in implementation; thereby, allowing the same software to run at different cost and performance.
- What is the power dissipation in CMOS IC technology?
- Suppose a new CPU has 85% of the capacitive load of the previous generation, 15% voltage reduction and 15% slower clock. What is the performance of the new machine in comparison to the old machine?
  - Answer: 0.52
- What is the difference between response time and throughput?
- What is the relationship between execution time and performance?
- Define the dif between clock period (cycle) and clock frequency (rate).
- What is the equation for a program execution time?
- A program runs on computer A with a 2 GHz clock in 10 s. What clock rate must computer B run at to run this program in 6 s? unfortunately, to accomplish this, computer B will require 1.2 times as many clock cycles as computer A to run the program.
  - Answer: 4 GHz
- What is the equation for the # of CPU clock cycles for a program?
- What is the equation for overall effective CPI
- Why do you use geometric mean?
  - First the execution times are normalized giving the "SPEC ratio". (the SPEC ratio is the inverse of execution time.)
  - Geometric mean helps with reproducibility
- What does Amdahl's Law measure?
  - The maximum expected improvement to overall system performance when only part of the system is improved
- What is the equation to calculate MIPS

Chapter 2

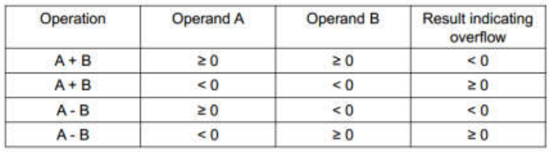
- What are the dif of RISC and CISC?
- What are the 3 key features that makes a RISC architecture fast?
  - Fix instructions size, small # of instruction formats, and fix opcode location
- Draw the 3 instruction formats and explain
- What does make the common case fast mean?
- How does increasing the # of read/write ports in a reg file impact speed?
  - Quadractically
- What kind of instruction format is lw and sw?
  - I format
- For lw and sw instructions, what are the restrictions on the offset number? And why?
  - Because the offset is a 2nd complement sign extend
  - For word addressing, +- 2^13 (8,192). For byte addressing +- 2^15 (32,768)
- What are the number of bits in a byte and word in this computer?
  - A byte is 8 bits and a word is 4 bytes
- What is alignment restriction and why does it apply to MIP 32?
  - Alignment restriction is the idea that it is more convenient to have everything byte addressable. This means that a word should cover a fix # of byte. In MIP 32, each chuck of data is 4 bytes long.
- What is little endian and big endian? And which one is MIPS? Also name what is ARM, Intel X86?
  - Big endian is when the left most byte is the most significant byte (the word address).
  - MIPS = big endian. ARM and Intel = little endian
- What are the ranges of a 16 bit negative #
  - + 2^15-1 to - 2^15
- Draw the instruction format for the command "sll $t1, $s0, 8
  - -- 0 -- unused -- $s0 -- $t2 -- shamt -- 0x00 --
- What are the instruction formats of instruction "and" and "andi"
- Draw the flow chart of the branch instruction and point out some of the key features

- 

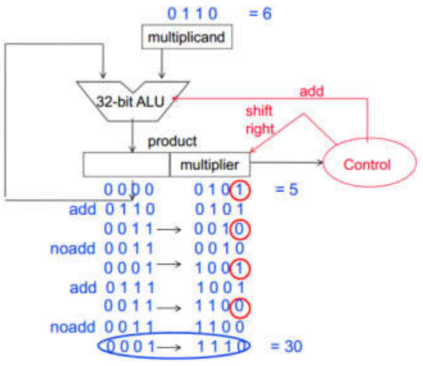
- What happen when a function needs more registers than the allocated argument and return registers?
  - Use the stack which grows from high address to low address
- What are the differences between a frame pointer and a stack pointer?
  - Frame pointer points to the beginning of the procedure (frame) whereas the stack pointer points to the latest space on the stack.
  - The SP can change during the function, but the FP cannot
- Draw the Stack, Dynamic Data (heap), static data (global variables) and your code in a memory hierarchy.
  - 

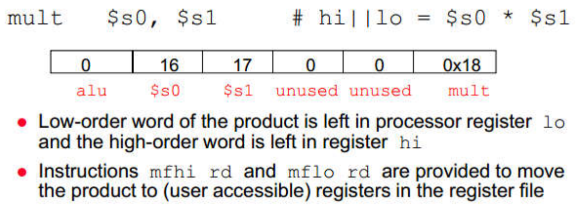
- What are race conditions?
  - When the result of a program can change depending on the the relative ordering of events

Chapter 3

- How to recognize an overflow?
  - 

Operation	Operand A	Operand B	Result indicating overflow
A + B	≥ 0	≥ 0	< 0
A + B	< 0	< 0	≥ 0
A - B	≥ 0	< 0	< 0
A - B	< 0	≥ 0	≥ 0

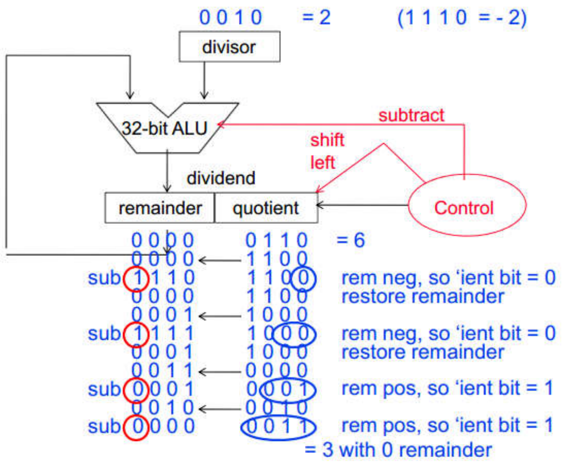
- What is the performance of RCA vs CLA
- Draw the add and shift right multiplier hardware
  - 

- How are the data of the multiplier stored?
  - 

```
mult   $s0, $s1     # hi||lo = $s0 * $s1
```

0	16	17	0	0	0x18
alu	$s0	$s1	unused	unused	mult

  - Low-order word of the product is left in processor register `lo` and the high-order word is left in register `hi`
  - Instructions `mfhi rd` and `mflo rd` are provided to move the product to (user accessible) registers in the register file
- Draw the left shift and subtract division hardware

- 
0 0 1 0 = 2 (1 1 1 0 = - 2)
divisor
32-bit ALU
subtract
shift left
dividend
remainder
quotient
Control
0 0 0 0 0 1 1 0 = 6
0 0 0 0 1 1 0 0
sub 1 1 1 0 1 1 0 0 rem neg, so 'ient bit = 0
0 0 0 0 1 1 0 0 restore remainder
0 0 0 1 1 0 0 0
sub 1 1 1 1 1 0 0 0 rem neg, so 'ient bit = 0
0 0 0 1 1 0 0 0 restore remainder
0 0 1 1 0 0 0 0
sub 0 0 0 1 0 0 0 1 rem pos, so 'ient bit = 1
0 0 1 0 0 0 1 0
sub 0 0 0 0 0 0 1 1 rem pos, so 'ient bit = 1
= 3 with 0 remainder
    - The key feature of this is to test whether or not, the subtract has a carry or not. If subtract without having to do a carry and the remainder is not negative then we have achieve our answer.
- How are the results of division stored?

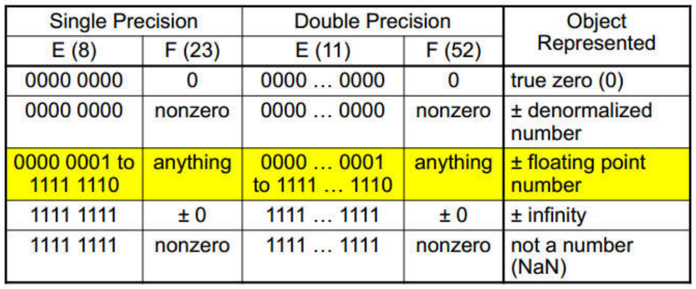
    -
      ```
      div    $s0, $s1        # lo = $s0 / $s1

                             # hi = $s0 mod $s1
      ```

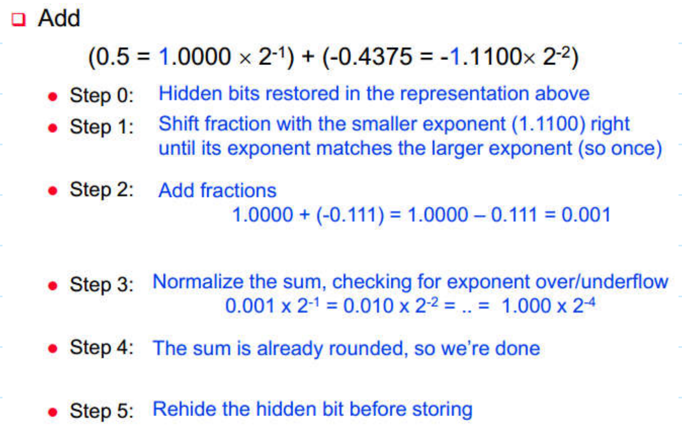
0	16	17	0	0	0x1A
alu	$s0	$s1	unused	unused	div

      - Instructions `mfhi rd` and `mflo rd` are provided to move the quotient and reminder to (user accessible) registers in the register file
- What is the format of the floating point representation?
    - ❑ Floating point representation $(-1)^{sign}$ x F x 2^E
      - **Still** have to fit everything in 32 bits (single precision)

s	E (exponent)	F (fraction)
1 bit	8 bits	23 bits
- What is overflow and underflow when it comes to floating point math
    - Overflow is when the exponent is too big to be represented by the bits. Underflow is when the exponent is too small to be represented by the bits
- What is the max of the sign magnitude representation?
    - +- |2^23 -1|
- For floating point, what is the format? What is the normalized signed magnitude? Hidden bit? Exponential bias?
    - Format: $(-1)^{sign} x\ (1+F) X\ 2^{E-bias}$
    - Normalized is when you make sure that the number has a leading 1 in the oneth place. And hidden bit is when you don't include this most significant 1 in the fractional part
    - Exponential Bias is when you add 127 to the exponent.
- How do you represent Smallest, zero and Largest + in exponential form?
    - Smallest+ = 1 x 2^{1-127} = 1 x 2^{-126} = ≈ 1.2 × 10^{-38}
      0 00000001 1.00000000000000000000000
    - Zero (true 0): 0 00000000 0.00000000000000000000000
      - Note that hardware detects true 0 (F and E all zeros) and does not restored the hidden bit of 1
    - Largest+ = (1 + 1-2^{-23}) x $2^{254-127}$ = (2 - 2^{-23}) x 2^{127} = ≈ 3.4 × 10^{+38}
      0 11111110 1.11111111111111111111111
    - 1.0_2 x 2^{-1} = (1+0) x $2^{126-127}$
      0 01111110 1.00000000000000000000000
    - 0.75_{10} x 2^4 = 0.11 x $2^{131-127}$
      0 10000011 0.11000000000000000000000
      $\Downarrow$
      0 10000010 1.10000000000000000000000
- How do you denote infinity and NaN in exponential form?

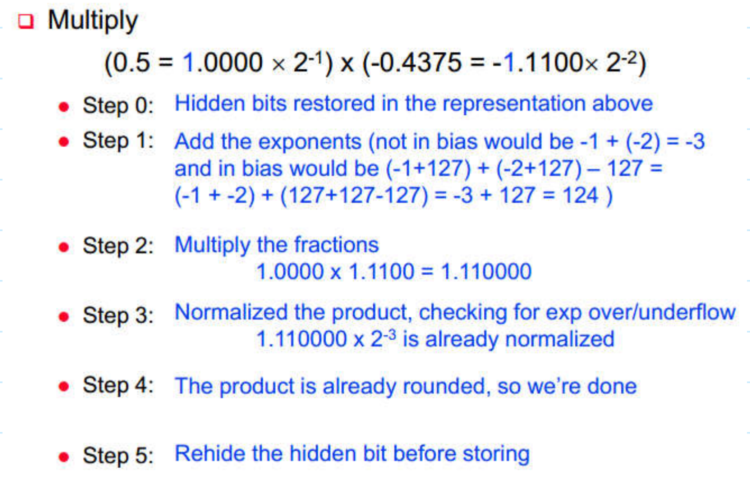
- 

Single Precision		Double Precision		Object Represented
E (8)	F (23)	E (11)	F (52)	
0000 0000	0	0000 … 0000	0	true zero (0)
0000 0000	nonzero	0000 … 0000	nonzero	± denormalized number
0000 0001 to 1111 1110	anything	0000 … 0001 to 1111 … 1110	anything	± floating point number
1111 1111	± 0	1111 … 1111	± 0	± infinity
1111 1111	nonzero	1111 … 1111	nonzero	not a number (NaN)

- Add 0.5 to -0.4375
  - 

    Add

    $(0.5 = 1.0000 \times 2^{-1}) + (-0.4375 = -1.1100 \times 2^{-2})$

    - Step 0: Hidden bits restored in the representation above
    - Step 1: Shift fraction with the smaller exponent (1.1100) right until its exponent matches the larger exponent (so once)
    - Step 2: Add fractions
      $1.0000 + (-0.111) = 1.0000 – 0.111 = 0.001$
    - Step 3: Normalize the sum, checking for exponent over/underflow
      $0.001 \times 2^{-1} = 0.010 \times 2^{-2} = .. = 1.000 \times 2^{-4}$
    - Step 4: The sum is already rounded, so we're done
    - Step 5: Rehide the hidden bit before storing
- Multiply 0.5 by -0.4375
  - 

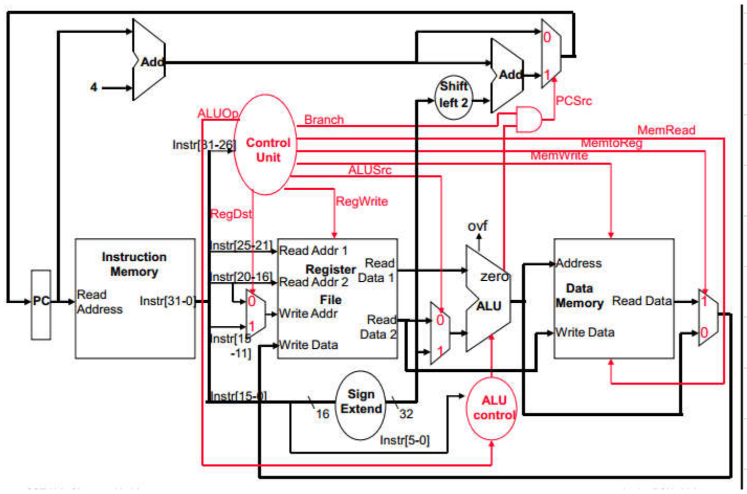
    Multiply

    $(0.5 = 1.0000 \times 2^{-1}) \text{ x } (-0.4375 = -1.1100 \times 2^{-2})$

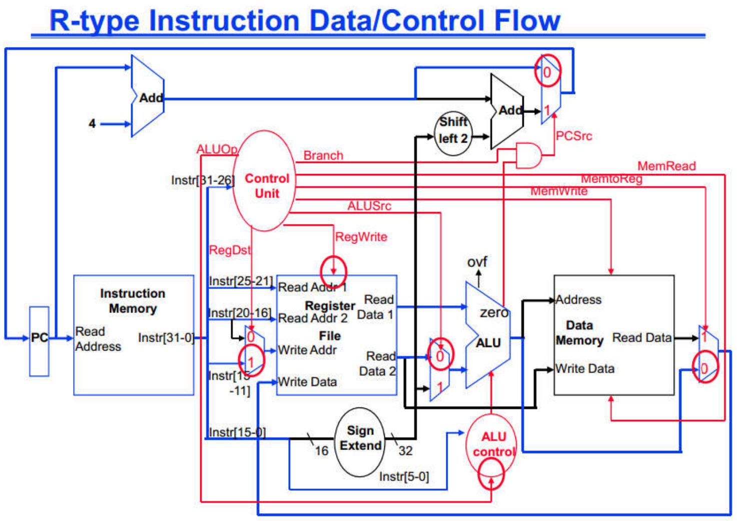
    - Step 0: Hidden bits restored in the representation above
    - Step 1: Add the exponents (not in bias would be -1 + (-2) = -3 and in bias would be (-1+127) + (-2+127) – 127 = (-1 + -2) + (127+127-127) = -3 + 127 = 124)
    - Step 2: Multiply the fractions
      $1.0000 \text{ x } 1.1100 = 1.110000$
    - Step 3: Normalized the product, checking for exp over/underflow
      $1.110000 \text{ x } 2^{-3}$ is already normalized
    - Step 4: The product is already rounded, so we're done
    - Step 5: Rehide the hidden bit before storing
- T/F → Shift left by one can always mean that the number was multiplied by 2? Explain
  - False. This is only true for unsigned number, in signed number, this can cause a positive number to become a negative number
- T/F → is floating point add/sub associative?
  - False

Chapter 4A:

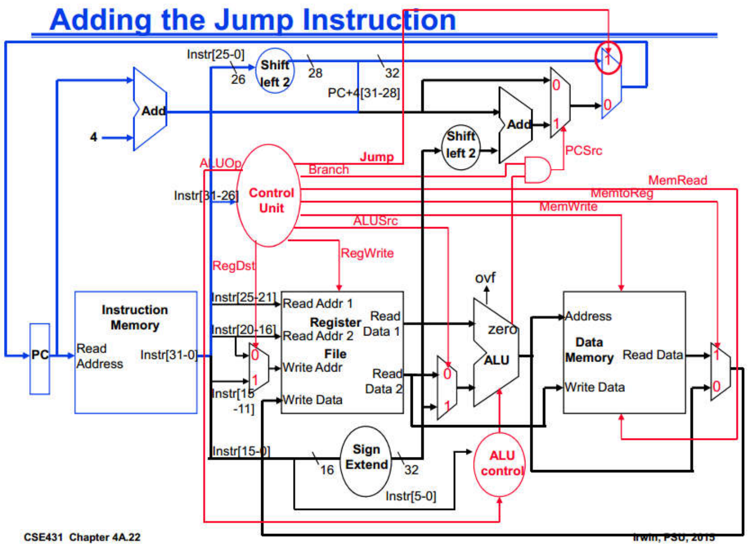
- Describe the single cycle design
  - The single cycle design is the design where no datapath resources can be used more than once in a single instruction cycle. This causes some datapath to be duplicated, but it allows for a better pipeline.
- Understand the diagram below

- 

- From the diagram above, describe the R-type instruction data flow.

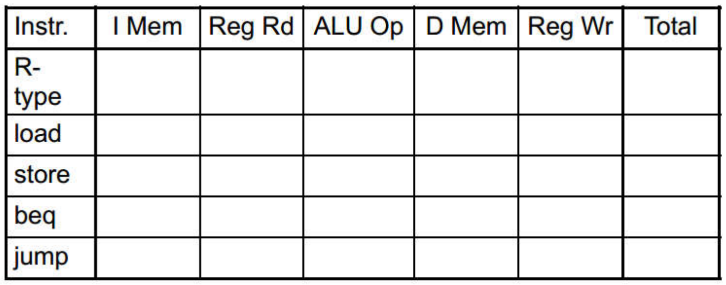
- 

- From the diagram above, describe the load instruction data flow
- From the diagram above, describe the branch instruction data flow
- From the diagram above, describe the jump instruction data flow

- 

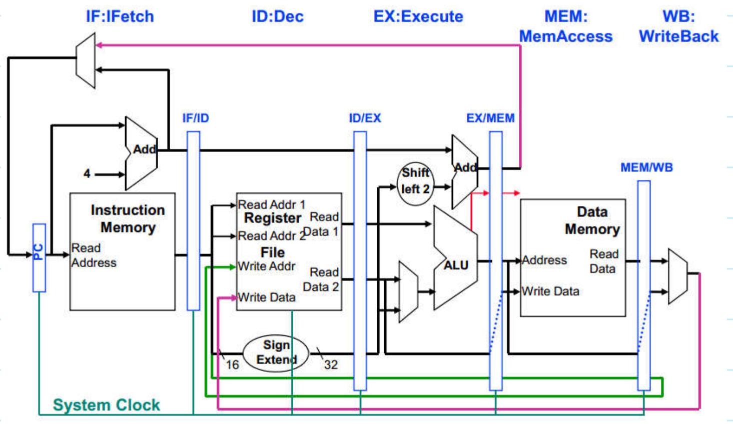
- Under ideal condition, what is the largest speedup in an n number of pipeline stages
    - n. Think about it, once the pipeline is full, each cycle would execute all n stages
- Does pipelining improve response time, throughput, or both?
    - Throughput
- Complete this table and explain why some cycles are wasted.

- Instruction and Data Memory (200 ps)
- ALU and adders (200 ps)
- Register File access (reads or writes) (100 ps)



Instr.	I Mem	Reg Rd	ALU Op	D Mem	Reg Wr	Total
R-type						
load						
store						
beq						
jump						

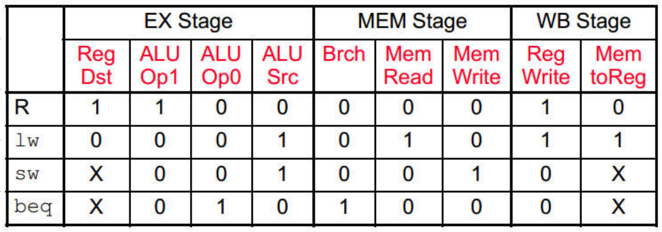
- What are the 5 advantages of the MIPS architecture in pipelining?
    - All instructions are the same length
    - Limited # of instruction format
    - Memory access only occur in a load or store
    - Each instruction writes at most one result
    - Operands must be aligned in memory so a single data transfer takes only one data memory access
- What is needed on top of the normal datapath drawing to enable pipelining?
    - State registers



CSE431 Chapter 4A.31 Irwin, PSU, 2015

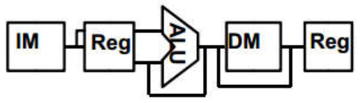
- Once a pipeline is full, what is the CPI

- IF Stage: read Instr Memory (always asserted) and write PC (on System Clock)
- ID Stage: decoding so no control signals ready yet

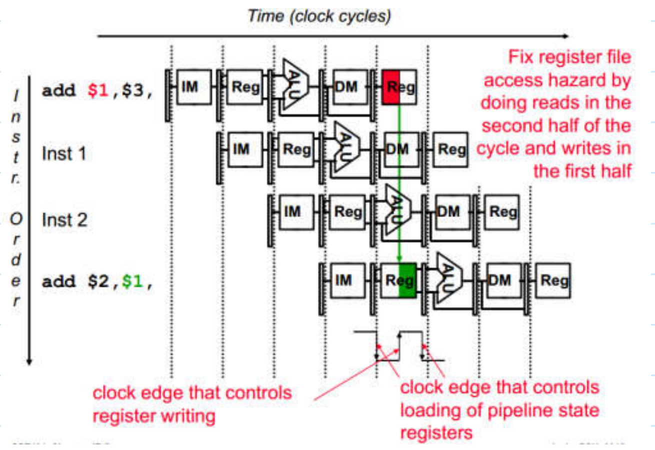
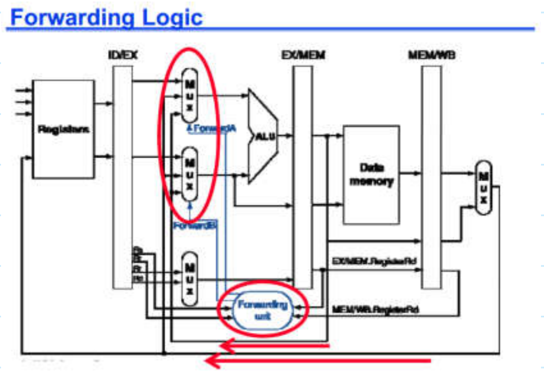


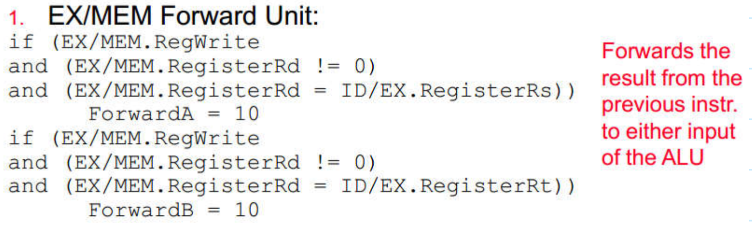
	EX Stage				MEM Stage			WB Stage	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Brch	Mem Read	Mem Write	Reg Write	Mem toReg
R	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

- What is the graphical representation of the MIPS pipeline?



Chapter 4B
- What is the hazard if you only have one memory instead of I$ and D$
    - You would have structural hazard as your 4th line down the pipeline would need to use the memory also
- Define Structural hazards, data hazards, and control hazards. Also give an example of each scenario
    - Structural hazards - when 2 instructions try to use the same resources at the same time

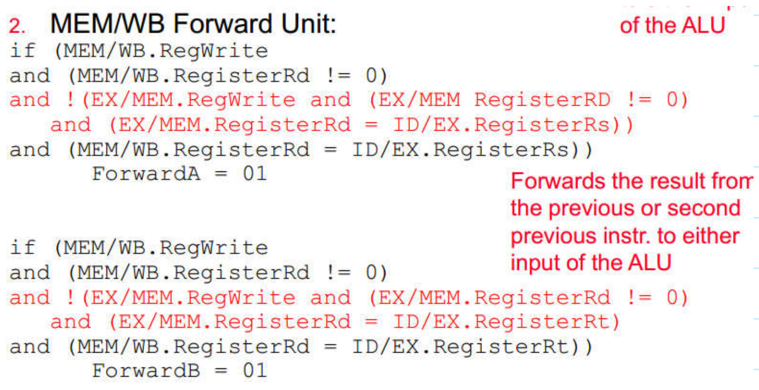
- 
Time (clock cycles)
Instr. Order
add $1,$3,
Inst 1
Inst 2
add $2,$1,
IM
Reg
ALU
DM
Reg
Fix register file access hazard by doing reads in the second half of the cycle and writes in the first half
clock edge that controls register writing
clock edge that controls loading of pipeline state registers
    - Data hazards - when an instruction try to use a data before it is ready
    - Control hazards - when an instruction try to make a decision for a control flow before the condition has been evaluated and the PC target is calculated.
- Define WAR, WAW, Load-useAlso give an example
- Explain what is data forwarding
    - Data is forwarded to the instruction that needs it rather than waiting for the end of the isntruction
    - 

- Write the code to determine if there is a data hazard and you need to do forwarding.
    - 

1. EX/MEM Forward Unit:

```
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd != 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
       ForwardA = 10
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd != 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
       ForwardB = 10
```

Forwards the result from the previous instr. to either input of the ALU

    - 

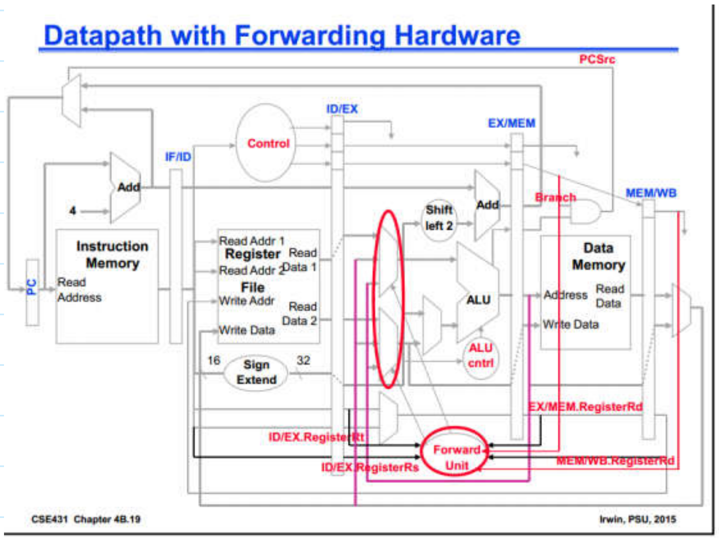
of the ALU

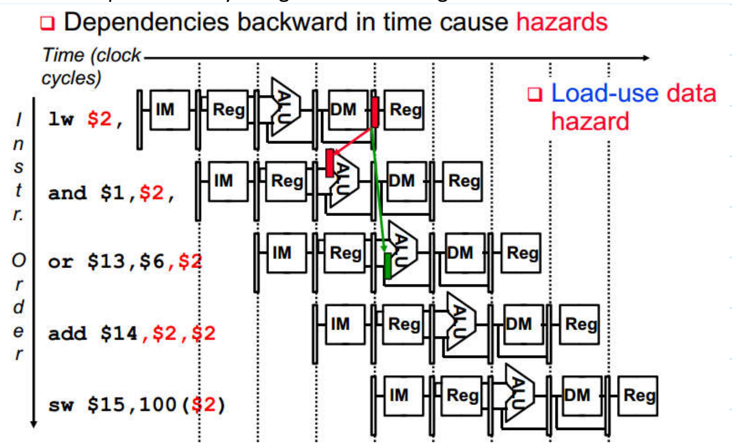
2. MEM/WB Forward Unit:

```
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd != 0)
and !(EX/MEM.RegWrite and (EX/MEM RegisterRD != 0)
   and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
       ForwardA = 01

if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd != 0)
and !(EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0)
   and (EX/MEM.RegisterRd = ID/EX.RegisterRt)
and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
       ForwardB = 01
```

Forwards the result from the previous or second previous instr. to either input of the ALU

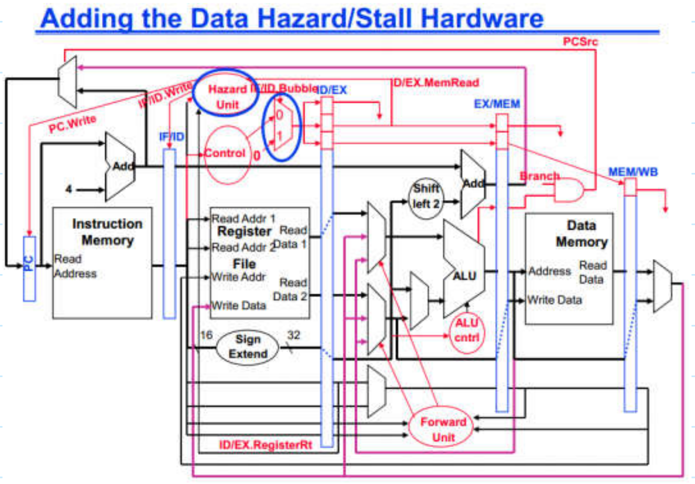
- 

- Data Hazard thus far
  - WAR - For example, sub follow by an "and"
  - Load-use
    - This can be prevented by using code reordering
    - 

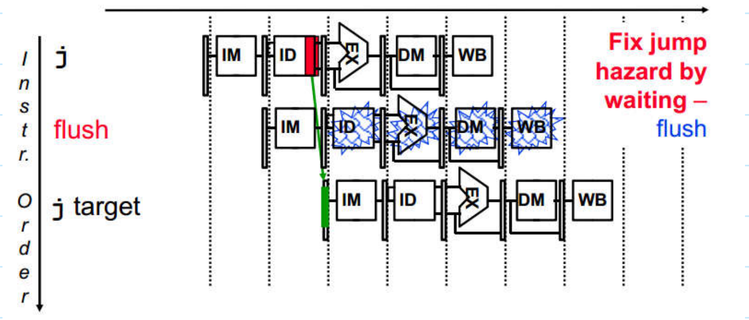
-
- Write the code to detect Load-use data hazard
  - 1. ID Hazard detection Unit:

```
if (ID/EX.MemRead
and ((ID/EX.RegisterRt = IF/ID.RegisterRs)
or   (ID/EX.RegisterRt = IF/ID.RegisterRt)))
stall the pipeline
```

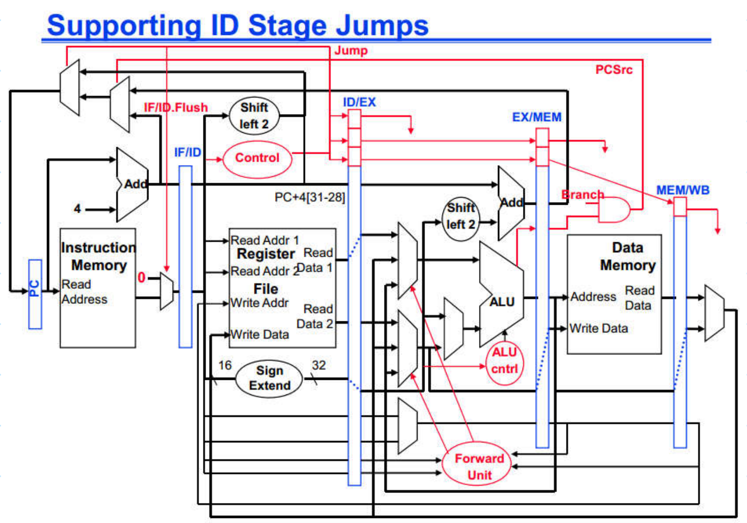
- What are the 3 things that you need to do inorder to implement a stall in the hardware?
  - Insert "bubble" b/w lw instruction (in the ex stage) and the load-use instruction (in the ID stage). Insert an IF/ID.Bubble
    - Set the control bits of EX, MEM, and WB Control fields of the ID/EX pipeline register to 0. The hazard control unit controls the mux that chooses b/w the real control values and the 0's
  - Prevent the IF and ID stages from proceeding down the pipeline. This can be achieve by preventing the PC and the IF/ID pipeline registers from changing
  - Let the lw instruction and the instructions after it in the pipeline (before it in the code) proceed normally down the pipeline.

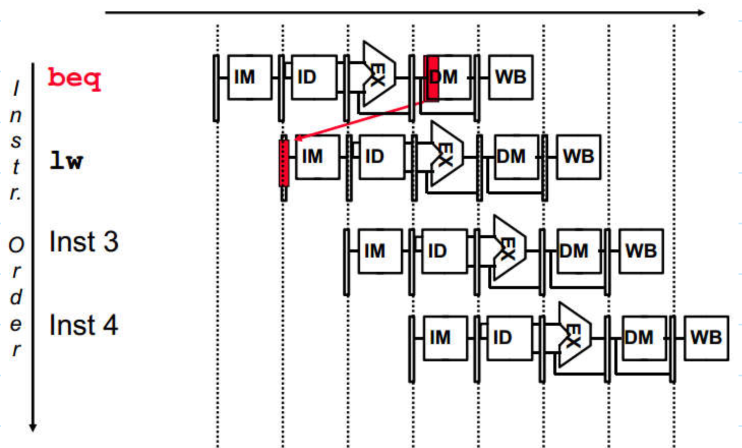
- 

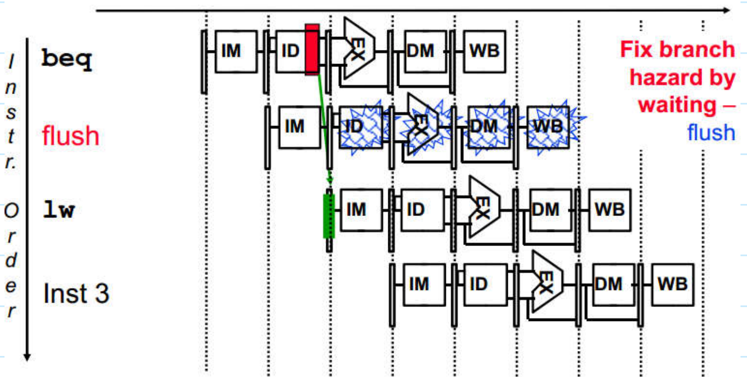
Chapter4C

- Define control hazard
  - When the following command is not sequential (aka PC + 4) and the flow of the pipeline is changed.
- What are the instructions that can cause a control hazard
  - Unconditional branch
  - Conditional branch
  - Exception
- What are 4 things that we can do you minimize the effect of control hazard?
  - Move the branch decision earlier in the pipeline
  - Guess and hope for the best
  - Stall
  - Delay decision
- If we can detect control hazard from a branch in the ID stage, how many stall(s) do we need?
  - 1
- What are the 2 types of stalls?
  - NOOP - Keep the instructions earlier in the pipeline from processing, insert NOOP by zeroing the control bits in the pipeline register at the appropriate stage, and let the instructions later in the pipeline progress normally down the pipeline
  - Flush - an instruction in the pipeline is replaced with a noop instruction. Key point here is that only one instruction.
    - 

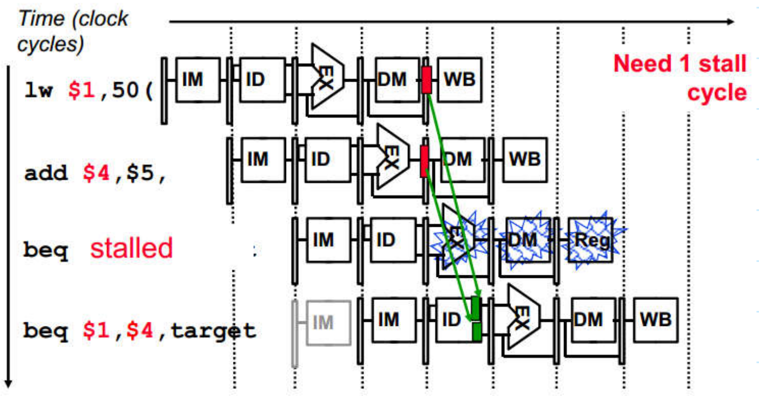
- Draw how you would implement an ID stage jump

- 

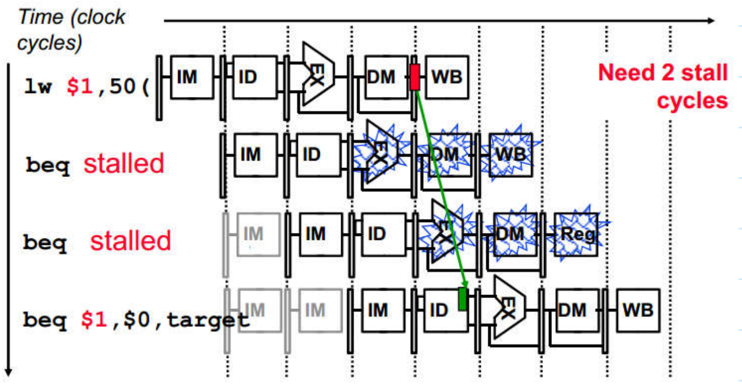
- Give an example where a branch instruction would cause a control hazard because you've jumped back in time? And how do you fix it.
  - 

  - You can fix it by flushing out all the instructions that are affected by the hazard
  - Move the branch decision hardware back to as early as possible
    - If you move it back to the EX stage then you limit it to 2 stalls
    - If you move it back to the ID stage then it would cause only 1 flush/stall
      - 

      - But now we need to add forwarding hardware in the ID stage
      - If a comparison register is a destination of the preceding ALU instruction or the 2nd preceding register then we must stall by one.

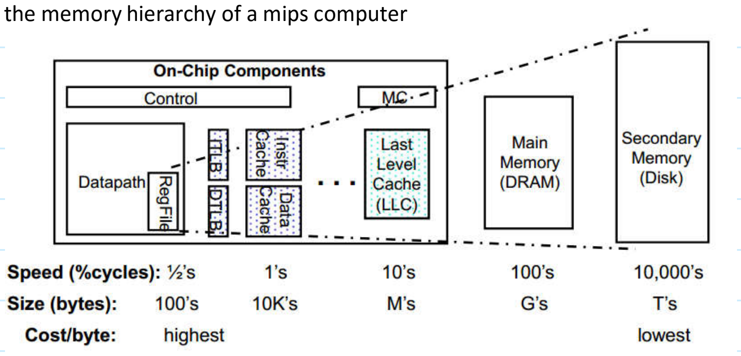
- 

    - If a comparison register is a destination of the preceding LOAD instruction then we must stall by 2

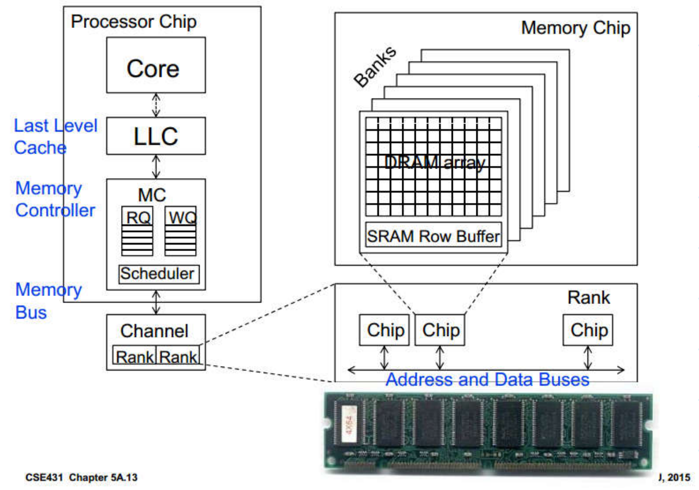
- 

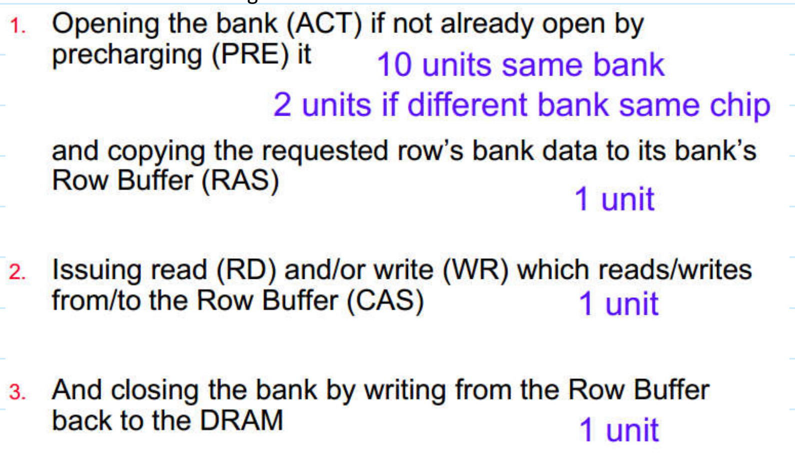
- What are 2 ways to reduce the delay of branches?
    - Move the branch decision back to the EX stage
        - This has 2 stalls
    - Move the branch decision back to the ID stage
        - This has 1 stall
- In normal hardware, how many stalls would a branch Always Taken cost?
    - 3 stalls if nothing was changed. 2 stalls if the branch hardware was moved back to EX stage. 1 stall if it was moved back to the ID stage
- How can you have it so that an always Taken branch would not cause any stall?
    - Use a BTB (Branch Target Buffer) in conjunction with the BHT.
        - The branch target instruction is fetched along with the sequential instruction in the fetch stage
- What is a BHT?
    - Branch History Table. It contains the lower order bits of the PC and it helps to tell whether or not the branch was taken last time
- Where on the loop does the NT prediction work best and why?
    - At the top of the loop because if it is at the bottom then it would guess wrong most of the time

Chapter 5A

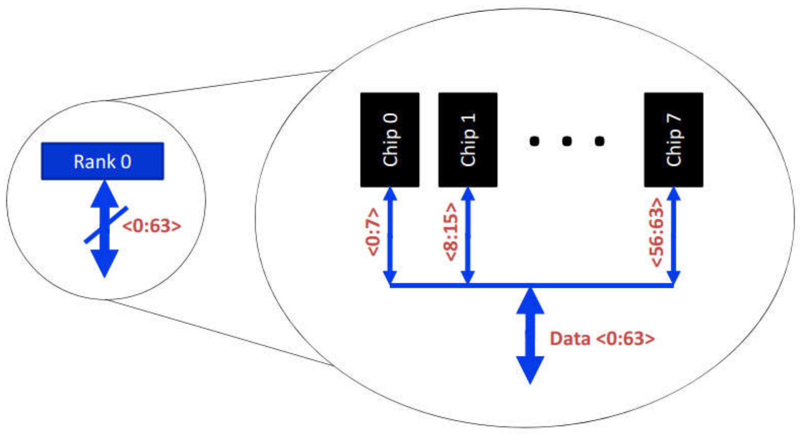
- Draw the memory hierarchy of a mips computer
    - 

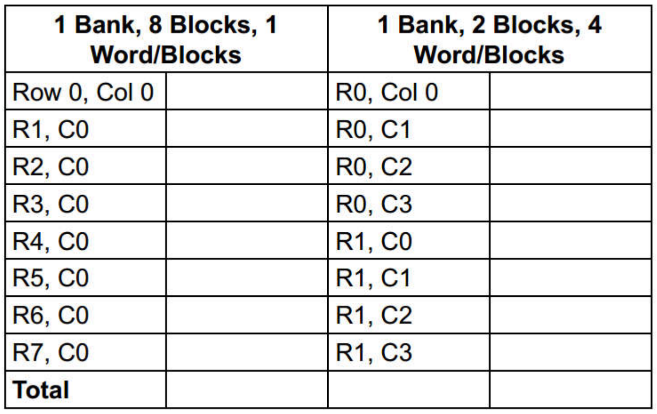
- What are the 2 localities? And how do they work?
    - Temporal - if a memory is referenced then it will likely be referenced again
    - Spatial - if a memory is referenced then the memory around it will likely be referenced
- Define block, hit rate, hit time, miss rate, miss penalty
    - Block - the minimal chunk of data in a cache that can be moved around in cache
    - Hit rate - the fractional of the data that were successful
    - Hit time - the time it takes to determine that it is a hit and retrieve the data
    - Miss rate - the factional of the data that were not successful

   - Miss penalty - the cost of the miss
- What type of RAM is used for cache and what type of RAM is used for main memory?
   - SRAM for cache and DRAM for main memory
- T/F SRAM is destruction and must be refreshed after read.
   - False, DRAM is destructive and must be refreshed after read
- Draw the memory structure from core to main memory DRAM
   - 

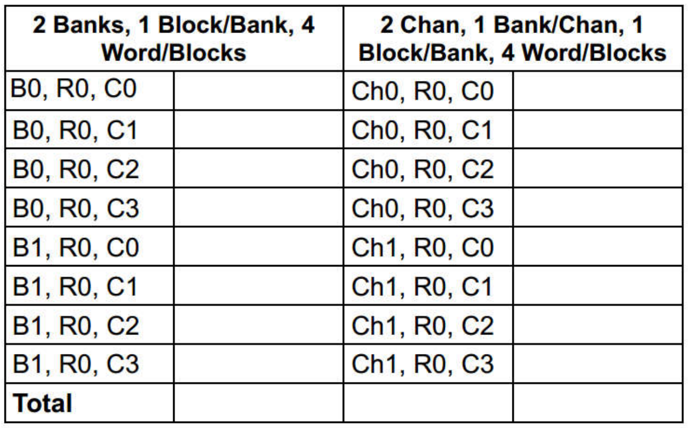
- List the DRAM subsystem organization in order from closest to process to furthest away
   - Channel, DIMM, Rank, chip, bank, row/column
- What type of memory is used for the Row buffer for the DRAM?
   - Sram
- What are RAS and CAS in RAM?
   - Row access strobe and column access strobe
- What are the 3 steps to accessed a "closed row"
   - Activate - The open the closed row
   - Read/write - reads/writes a column in the row buffer
   - Precharge - closes the row and prepares the DRAM bank for the next access
- What are the bank access timing of DRAM?
   - 

1. Opening the bank (ACT) if not already open by precharging (PRE) it 10 units same bank
   2 units if different bank same chip
   and copying the requested row's bank data to its bank's Row Buffer (RAS) 1 unit
2. Issuing read (RD) and/or write (WR) which reads/writes from/to the Row Buffer (CAS) 1 unit
3. And closing the bank by writing from the Row Buffer back to the DRAM 1 unit

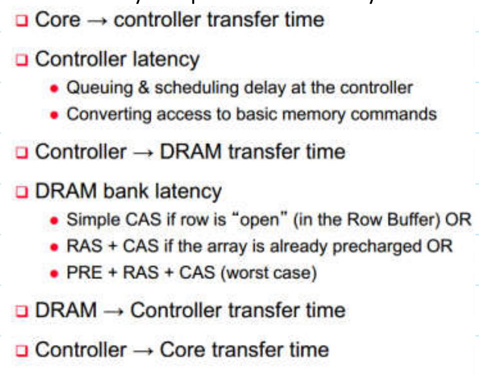
- Draw the breakdown of rank
   - 

- For DIMM, how does it select which rank to get data from?
    - CS line help with the rank selection
- How does the memory controller maximizes the bandwidth of the memory for DRAM
    - Exploit row buffer locality as much as possible
    - Reordering burst to avoid bank conflicts
    - Prioritizing read over writes
    - Preferring read after read and write after write
- Fill this timing table for DRAM out.
    - 

1 Bank, 8 Blocks, 1 Word/Blocks		**1 Bank, 2 Blocks, 4 Word/Blocks**	
Row 0, Col 0		R0, Col 0	
R1, C0		R0, C1	
R2, C0		R0, C2	
R3, C0		R0, C3	
R4, C0		R1, C0	
R5, C0		R1, C1	
R6, C0		R1, C2	
R7, C0		R1, C3	
Total			

    - 

2 Banks, 1 Block/Bank, 4 Word/Blocks		**2 Chan, 1 Bank/Chan, 1 Block/Bank, 4 Word/Blocks**	
B0, R0, C0		Ch0, R0, C0	
B0, R0, C1		Ch0, R0, C1	
B0, R0, C2		Ch0, R0, C2	
B0, R0, C3		Ch0, R0, C3	
B1, R0, C0		Ch1, R0, C0	
B1, R0, C1		Ch1, R0, C1	
B1, R0, C2		Ch1, R0, C2	
B1, R0, C3		Ch1, R0, C3	
Total			

- What are the latency components in memory?
    - 

❑ Core → controller transfer time

❑ Controller latency
- Queuing & scheduling delay at the controller
- Converting access to basic memory commands

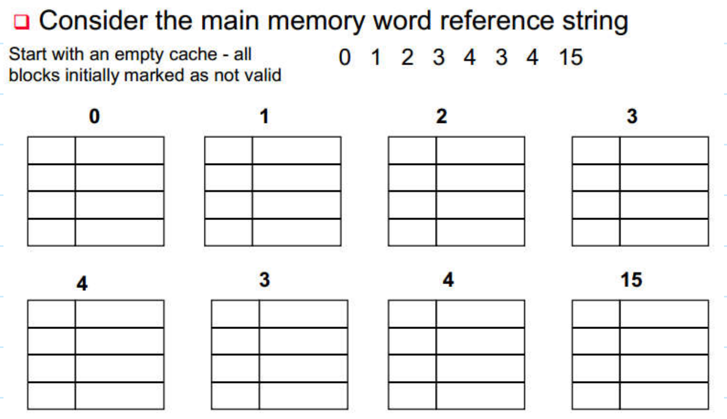
❑ Controller → DRAM transfer time

❑ DRAM bank latency
- Simple CAS if row is "open" (in the Row Buffer) OR
- RAS + CAS if the array is already precharged OR
- PRE + RAS + CAS (worst case)

❑ DRAM → Controller transfer time

❑ Controller → Core transfer time

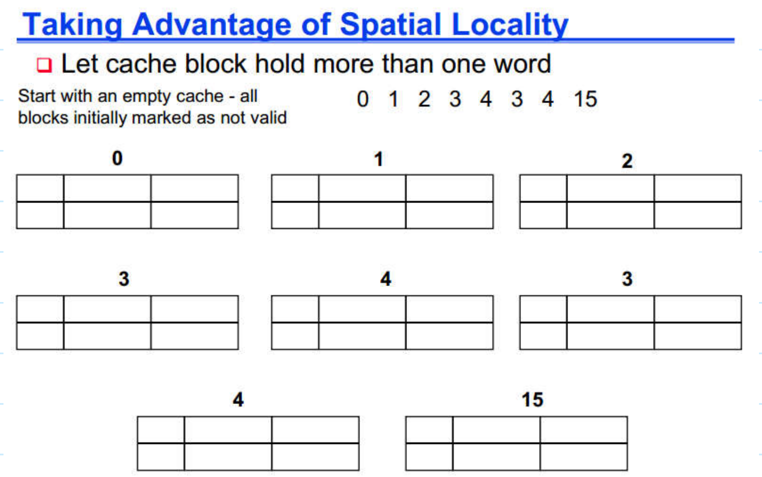
Chapter 5B
- What is direct mapped cache?
- What is the equation for finding the block address of a direct mapped cache.
    - (block address) mod (# of blocks in the cache)
- Using Direct mapped, finish the table below

- 

  Consider the main memory word reference string

  Start with an empty cache - all blocks initially marked as not valid

  0 1 2 3 4 3 4 15

- 

  Taking Advantage of Spatial Locality

  Let cache block hold more than one word

  Start with an empty cache - all blocks initially marked as not valid

  0 1 2 3 4 3 4 15

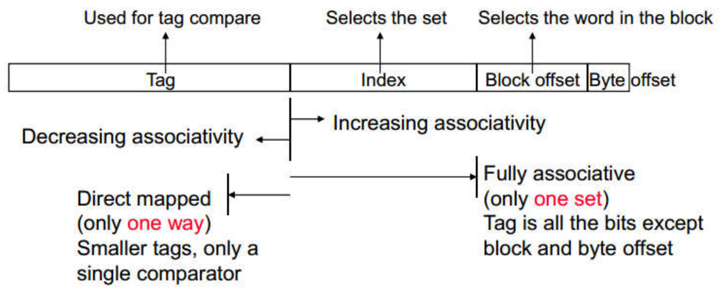
- Will miss rate increases if the block size decreases in respect to the cache size? Why?
  - Yes, think of capacity conflicts
- Can write hits occur for both I$ and D$?
  - No, it can only occur for D$
- What are the differences between consistent and inconsistent cache and memory?
  - Consistent would require write though
  - Inconsistent can allow write back. We just need to mark the block of data with a dirty bit.
- What are the 3 types of cache misses?
  - Compulsory - cold start misses
  - Capacity - cache cannot contain all blocks accessed by the program
  - Conflict - multiple memory locations mapped to the same location
- Can read misses happens to both I$ and D$? What happen when it does happen?
  - Yes, when it does happen, the processor need to stall the pipeline. The data is then fetch from lower level memory and place into the higher level. This may require the processor to evict a word (which may requires write back if it is dirty). Finally, the data from the higher level cache is then pass to the core.
- Can write misses happens to both I$ and D$? And what happen when it does happen? And what are the different actions we can take?
  - No, it can only happen to the D$ as I$ does not have write access
  - When it does happen, the data word is fetch from the lower level memory. This may require the processor to evict a dirty data word, in which case, a write back is required. Once the data is back into the higher cache we can take one of the 2 approaches
    - Write to the cache to the cache and set it as dirty
    - Write to the write buffer and invalid the word in the cache
- What is the equation for read-stall cycles and write stall cycles
  - $Read\ stall\ cycles = \frac{reads}{program} x\ read\ miss\ rate\ x\ read\ miss\ penalty$
  - $Write\ stall\ cycles = \frac{writes}{program} x\ miss\ rate\ x\ miss\ penalty$
  - Memory stall cycles = read stall cycles + write stall cycles
- What does AMAT stand for and what is the equation for it?
  - $AMAT = time\ for\ a\ hit + miss\ rate\ * miss\ penalty$

- What is the AMAT for a core with a 20 psec clock, a miss penalty of 50 clock cycles, a miss rate of 0.02 misses per instruction and a cache access time of 1 clock cycle?

  AMAT = 1 + 0.02x50 = 2 cycles

- How do you calculate the block address for n-way set associative cache?
  - (block address) mod (# sets in the cache)

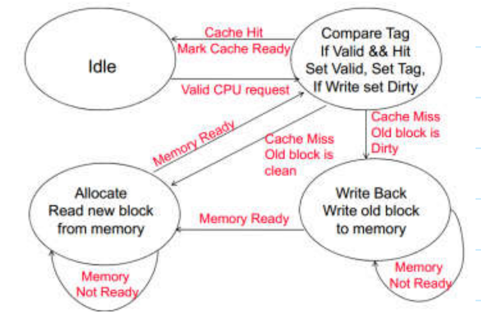
  ❑ For a **fixed size** cache, each increase by a factor of two in associativity doubles the number of ways (i.e., doubles the number of blocks per set) and halves the number of sets – decreases the size of the index by 1 bit and increases the size of the tag by 1 bit

- 

- How do we pick which block in the set to replace?
  - LRU - Least recently used algorithm
- Ways of reducing caches miss rate
  - Increase associativity
  - Use multiple levels of caches
  - Hardware prefetching
- Calculate the CPIstalls for CPI_{ideal} of 2, 100 cycles miss penalty (to main memory) and a 25 cycle miss penalty (to UL2$), 36% load/stores, a 2% (4%) L1I$ (L1D$) miss rate, add a 0.5% UL2$ miss rate
  - $$CPI_{stalls} = 2 + .02\times25 + .36\times.04\times25 + .005\times100 + .36\times.005\times100 = 3.54$$

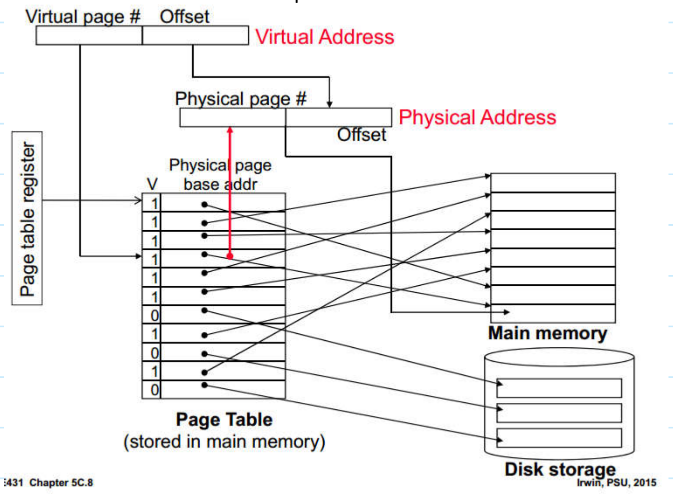
    (as compared to 5.44 with no L2$)
- What are the design consideration differences b/w L1 and L2 caches?
  - L1 is to minimize hit time, L2, minimize miss rate
- What does L2 hit time determines and what does L2 miss rate determines?
  - L2 hit time determines L1 miss penalty
  - L2 miss rate determines the global miss rate
- Compare the of Split vs unified caches
  - Split caches
    - Help prevent structural hazard (
  - Unified caches
    - Minimizes miss rate, fewer capacity misses (unused instr. Space can be used for data), more conflict misses
- Explain hardware prefetching
  - Hardware prefetching is when you fetch data for the cache proactively. The key to predict the next miss addresses accurately.
  - It must relies on unused processor bandwidth
- What is a simple case of hardware prefetching
  - A simple case of this would be where you fetch the instruction that comes after the current instruction. This exploit spacial locality
- What is the case where hardware prefetching is bad
  - If the case the fetch is wrong and it is replacing that we actually need; thereby increasing the miss rate of the program

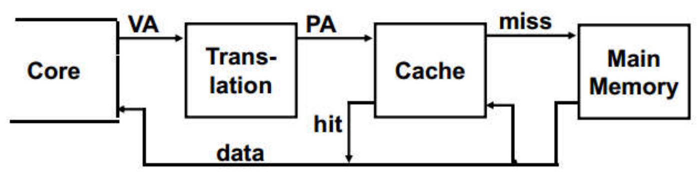
Four State Cache Controller

- 

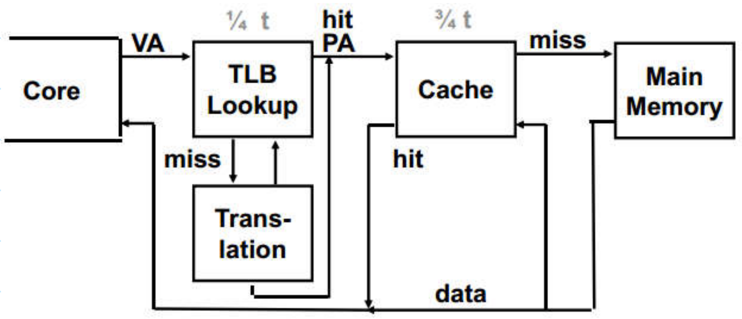
- What is code transformation?
    - This is where you change the data access pattern to influence miss rate and hit rate
- What is loop fusion?
    - This is where you merge 2 loops together to become one. This reduces cost of test and branching
    - This exploits temporal locality
- What is loop interchange?
    - This is where you just swap the 2 loops so that you can achieve better spacial locality. Aka. Aligning the direction of the access with the way the memory blocks are organized.
- Summary: What are 3 ways that you can improve cache performance?
    - Reduce the hit time of the cache
    - Reduce the miss rate of the cache
    - Reduce miss penalty

Chapter5C

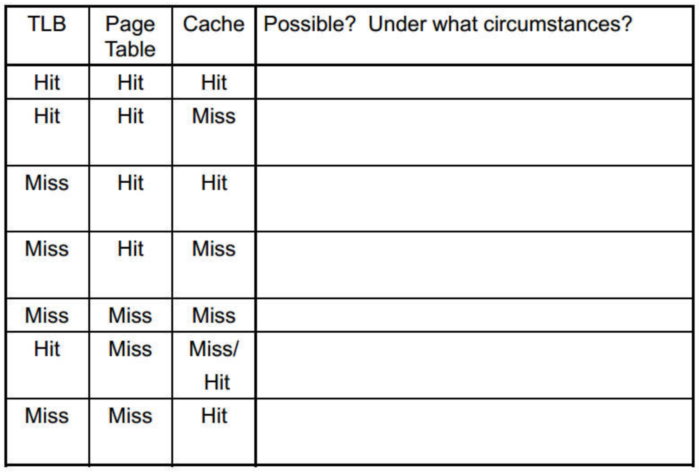
- What are the 3 main concepts of virtual memory?
    - Use main memory as cache for the secondary memory
        - This allows for efficient and safe sharing of memory between multiple programs/processes. This also allows for the computer to run programs that are larger than main memory
    - The core and the os work together to translate the virtual address to physical addresses
    - Exploiting the principle of locality
        - Programs tend to only use small portion of their address space over long portion of their execution time
- What are pages in a virtual memory and how is it different from segment?
    - Pages a the smallest address chunk.
    - Pages are fixed in size; whereas, segments are not.
- Where does the page table resides?
    - In main memory
- Draw the virtual address translation process
    - 

- Draw the flow chart for the virtual address translation process
    - 

- What is the TLB?
    - TLB stands for Translation Lookaside Buffer. It is a small cache that help in the translation process. With the TLB, very memory access does not require 2 accesses of the main memory now.
- What are the differences between the TLB and the PT?
    - TLB is a cache whereas PT is just a memory space
- Draw the flow chart of the translation process with TLB in it.

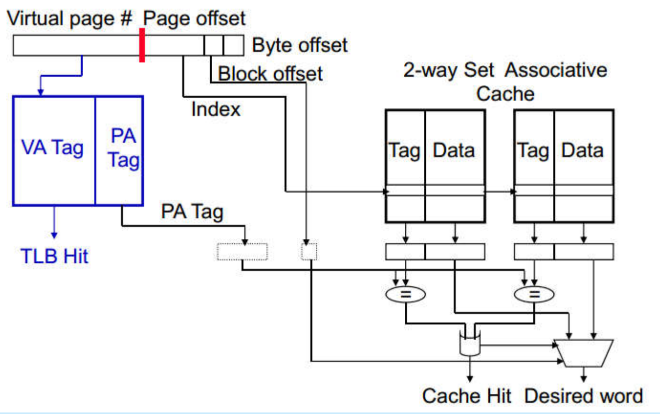


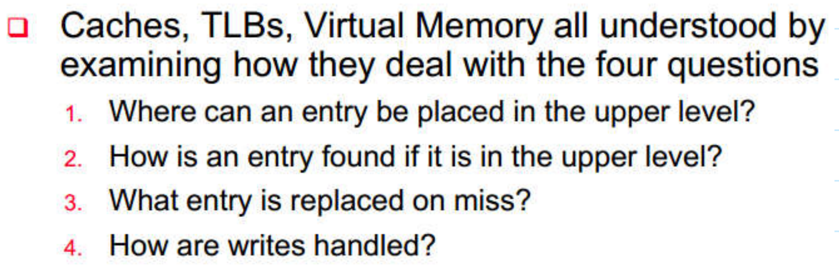
- What 2 things can a TLB miss indicate?
  - One, it means that the data is located in the PT and need to be fetch.
  - Two, a page fault
- Fill the table below



TLB	Page Table	Cache	Possible? Under what circumstances?
Hit	Hit	Hit	
Hit	Hit	Miss	
Miss	Hit	Hit	
Miss	Hit	Miss	
Miss	Miss	Miss	
Hit	Miss	Miss/ Hit	
Miss	Miss	Hit	

- What is the problem with virtual addressed cache?
  - 2 programs can have 2 different virtual addresses pointing to the same physical address. This is call aliasing, which can lead to coherence issues
- What does it means to have a overlap cache access with the tlb access?
  - Works when the high order bits of the VA are used to access the TLB while the low order bits are used as index into cache





- ❑ Caches, TLBs, Virtual Memory all understood by examining how they deal with the four questions
  1. Where can an entry be placed in the upper level?
  2. How is an entry found if it is in the upper level?
  3. What entry is replaced on miss?
  4. How are writes handled?