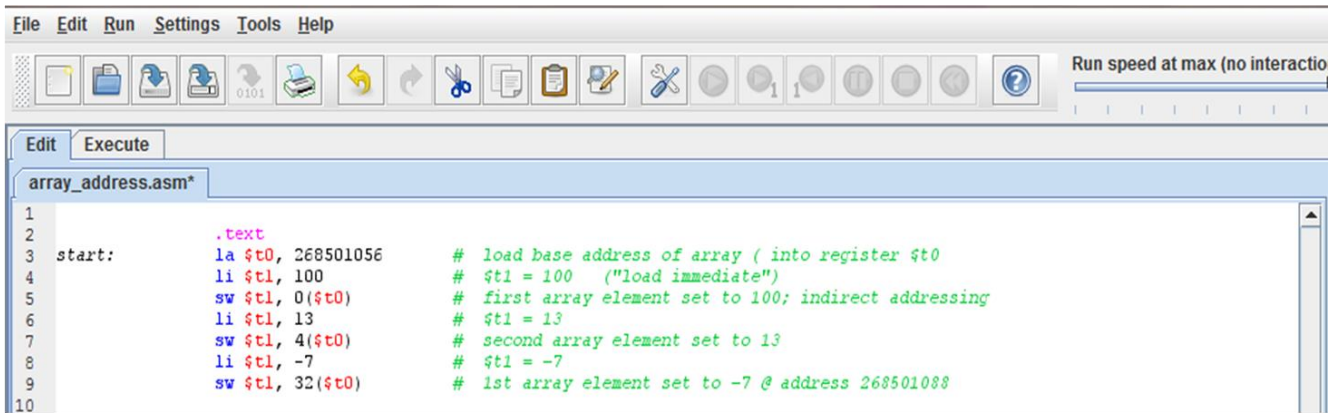
Final Exam – Study Guide

- About 20 T/F questions.

e.g. - Associativity and multi-level caching are two methods that can be used to improve cache performance.
- Load Word lw is an R-type MIPS instruction.
- First-level caches are more concerned about hit time.
- Increasing the CPI for a program, enhances performance.

- Be able to solve problem related to processor performance.
- e.g. Consider two different implementations, M1 and M2 of the same ISA. M1 has a clock rate of 1.5 GHz and M2 has a clock rate of 2.5 GHz. There are four instruction classes in the ISA. Load 25%, stores 15%, R-type 45%, Branches 20%....

- Upon executing the following MIPS code, update the memory array by penciling in the appropriate values in the table shown below.

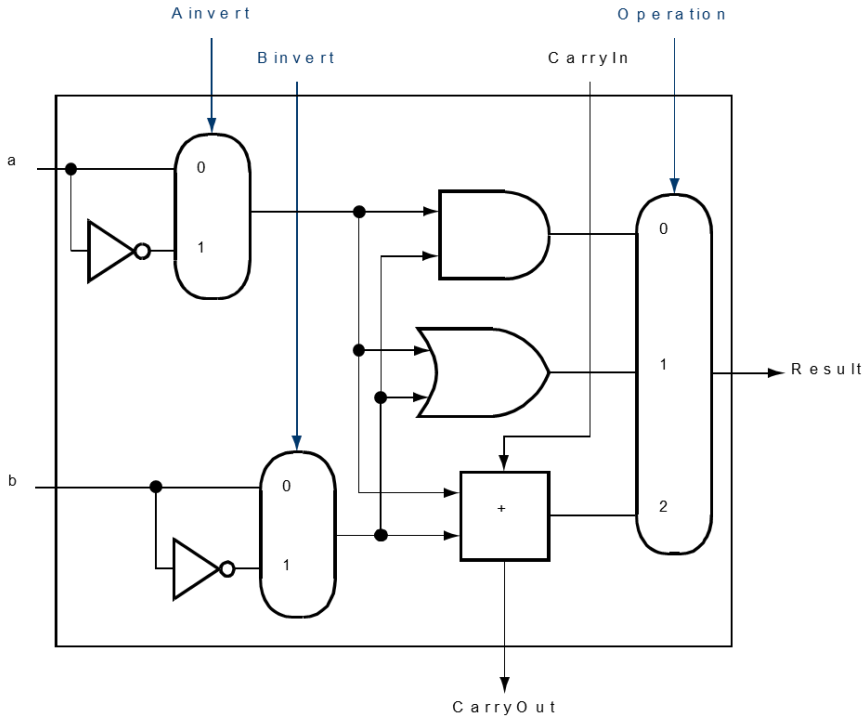


Data Segment

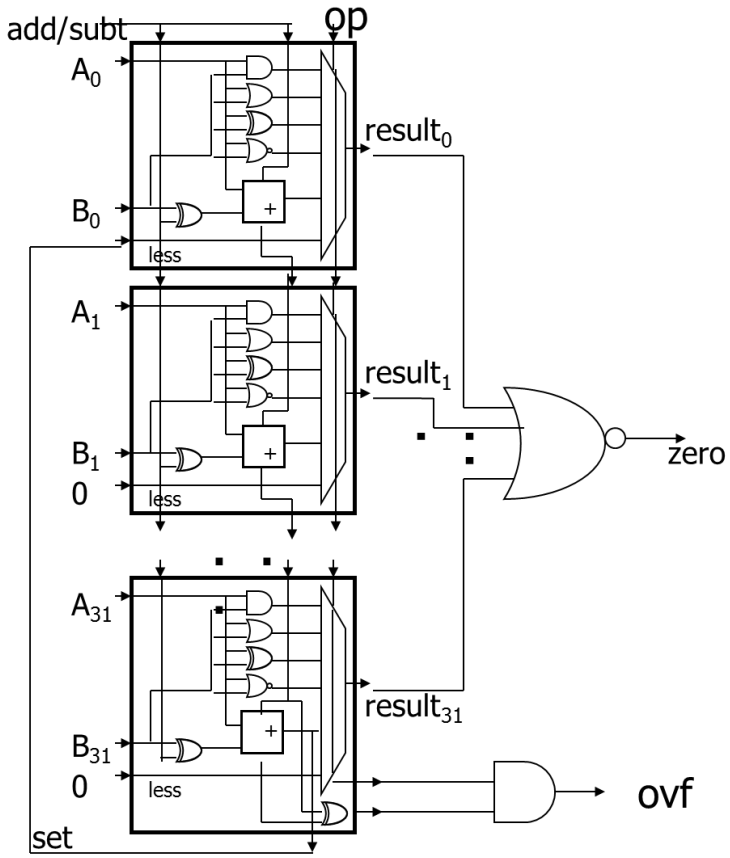
Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	0	0	0	0	0	0	0	0
268501024	0	0	0	0	0	0	0	0
268501056	100	13	0	0	0	0	0	0
268501088	-7	0	0	0	0	0	0	0
268501120	0	0	0	0	0	0	0	0
268501152	0	0	0	0	0	0	0	0
268501184	0	0	0	0	0	0	0	0
268501216	0	0	0	0	0	0	0	0
268501248	0	0	0	0	0	0	0	0
268501280	0	0	0	0	0	0	0	0
268501312	0	0	0	0	0	0	0	0
268501344	0	0	0	0	0	0	0	0
268501376	0	0	0	0	0	0	0	0

0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

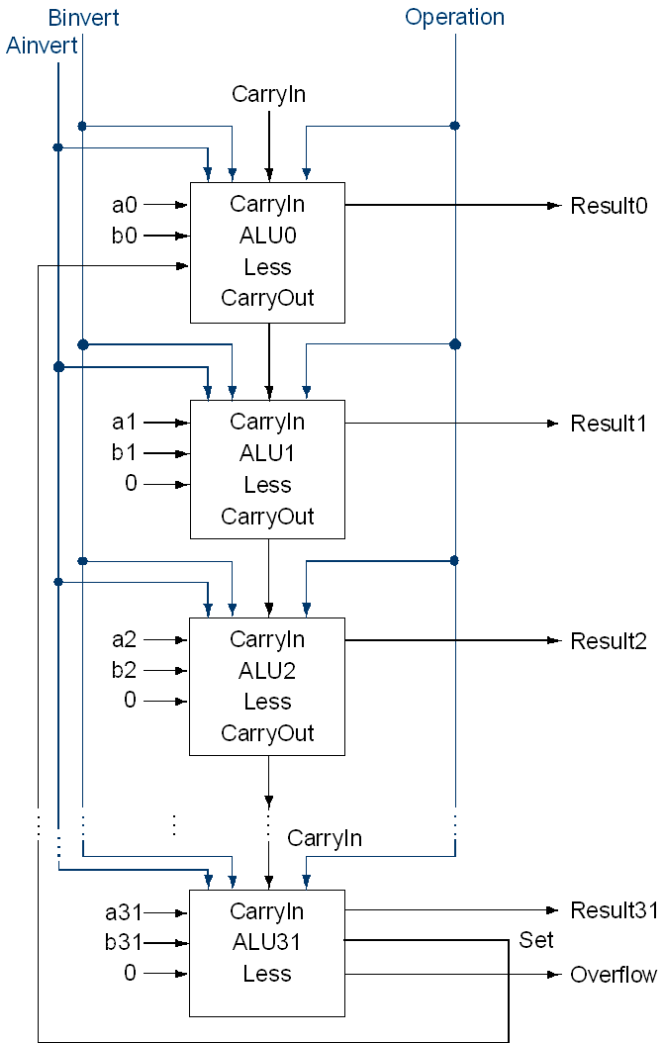
- Review ALU operation



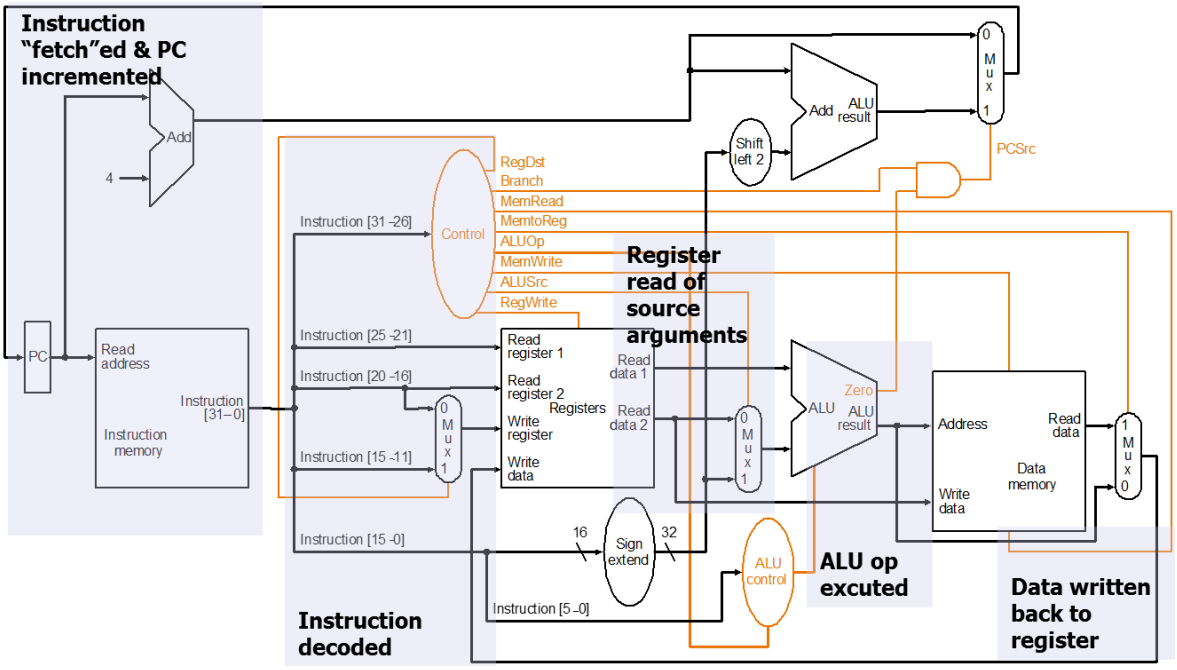
- Tailoring ALU to support beq instruction



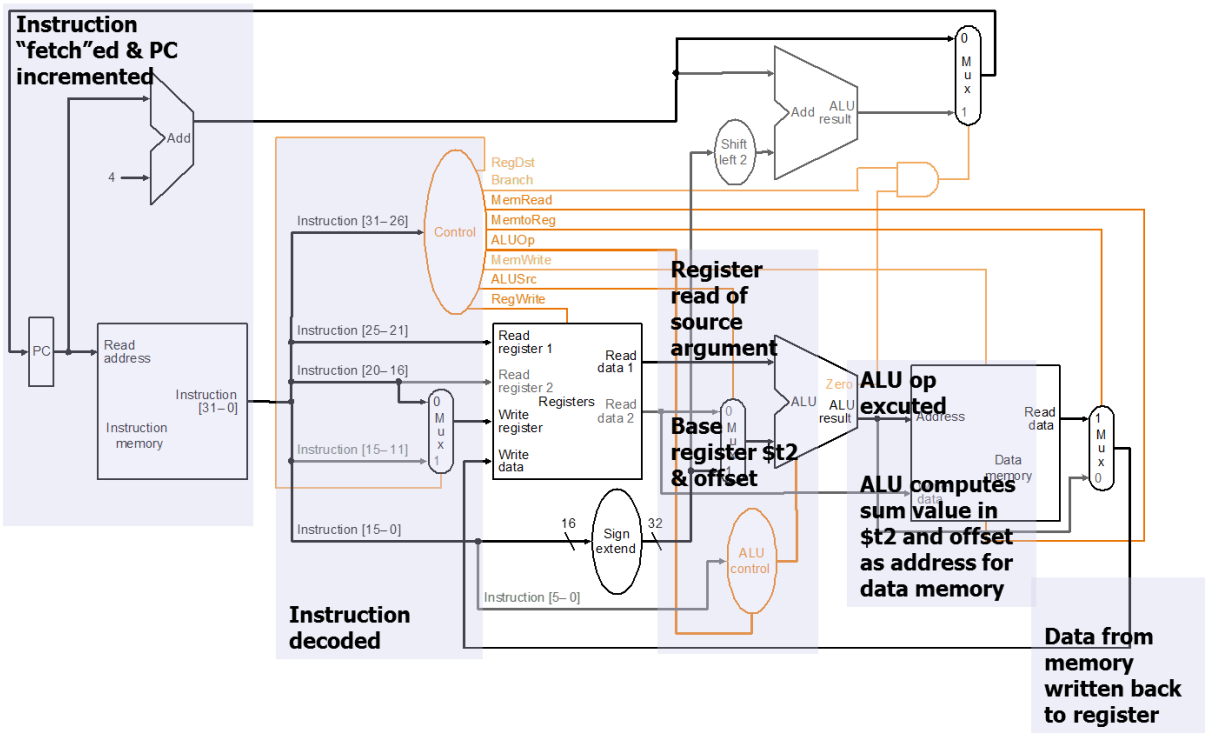
- Tailoring ALU to support slt instruction

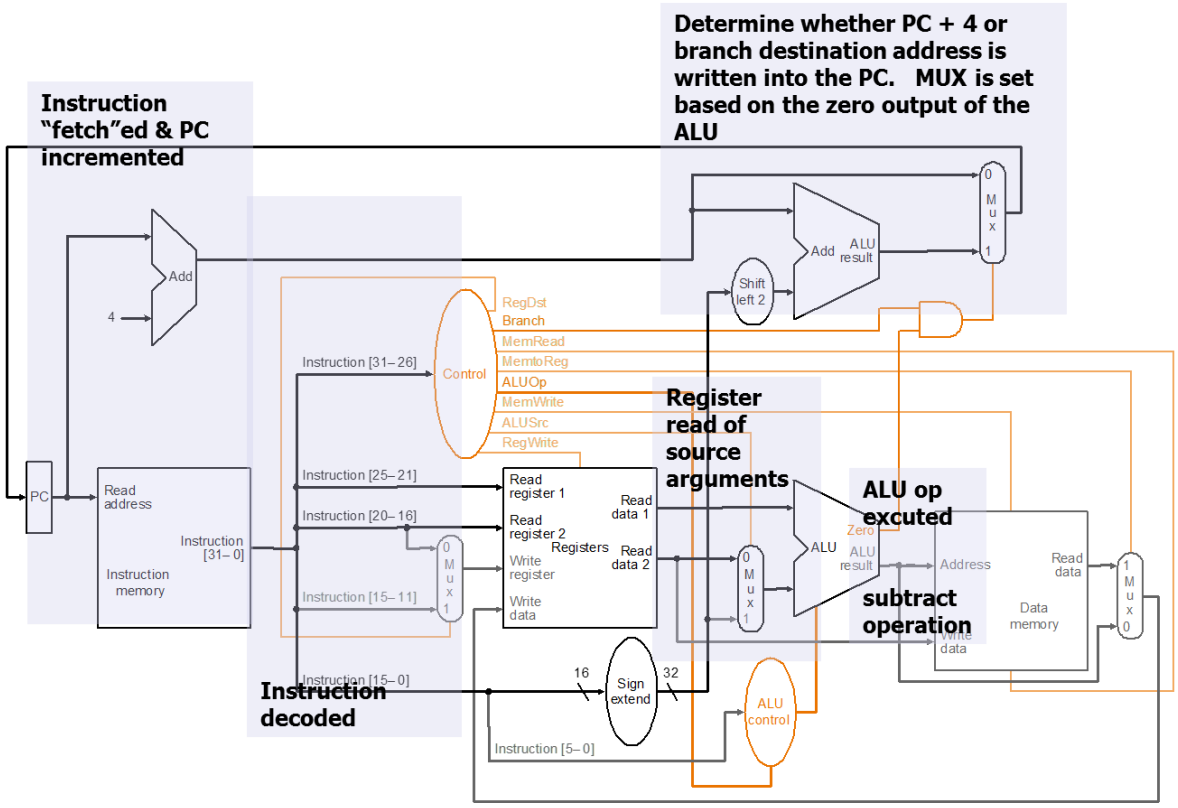


- Identify phases of R-type, I-type (load) and branch MIPS instructions:
-



lw $t1, offset($t2)





- *Pipelining is an implementation technique in which multiple instructions are overlapped in execution. Today, pipelining is the key to making processors fast.*
- Data hazards: attempt to use item before it is ready instruction depends on result of prior instruction still in pipeline. Dependencies that "go backward in time" are data hazards

- Identify the hazards in the following code.

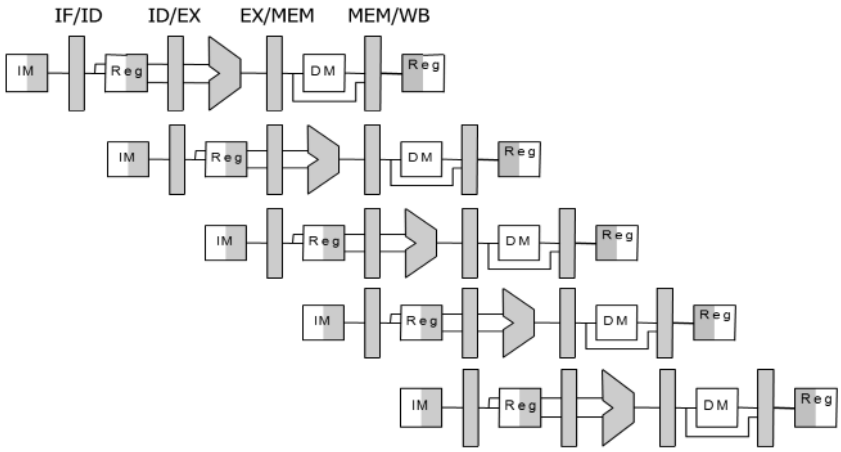
```
sub     $2,  $1, $3
and     $12, $2, $5
or      $13, $6, $2
add     $14, $2, $2
sw      $15, 100($2)
```

Fix the hazards using NOPs (software solution.)

- Fix the hazards by show the forwarding paths needed to execute the following four instructions:

```
add $3, $4, $6
sub $5, $3, $2
lw $7, 100($5)
add $8, $7, $2
```

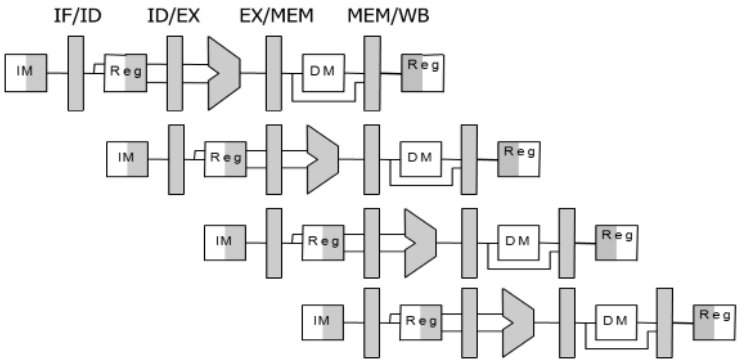
Use the diagram below. Rewrite the code



- Consider executing the following code on a pipelined datapath.

```
lw $4, 100($2)
sub $6, $4, $3
add $2, $3, $5
```

Show any 'bubbles for stalls' and/or forwarding paths needed to resolve the hazards. Use the diagram below.



- Compare the performance for single-cycle, multi-cycle, and pipeline control using the SPECint2000 instruction mix
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% ALU

The number of clock cycles for each instruction class:

- Loads: 5
- Stores: 4
- Branches: 3
- Jumps: 3
- ALU: 4

Start with performance of single-cycle machine:

- 200 ps for memory access
- 100 ps for ALU operation
- 50 ps for register file read or write

(a) What is the clock cycle time for single-cycle datapath?

(b) What is the average CPI for the multiple cycle design?
(c) What is the average CPI for the pipeline design?
(d) Find the average instruction time for single-cycle, multicycle and pipelined designs.

Loads, stores, and ALU take 1 clock cycle. Branches take 1 clock cycles when predicted correctly and 2 when not. Jump CPI = 2. Note that the long cycle time of memory is a performance bottleneck for pipelined and multicycle design.

HW chapter 5

1. Assume an instruction cache miss rate for a program is 2% and a data cache miss rate is 4%. If the processor has a CPI of 2 without any memory stalls and the miss penalty is 100 cycles for all misses, determine how much faster a processor would run with a perfect cache that never misses. The frequency of all loads and stores is 36% for SPECint2000

   Solution:
   Assume instruction count = I
   # of memory miss cycles for instructions =
   Instruction cycle miss = I x 2% x 100 cycles = 2.00 x I
   # memory miss cycles for data references
   Data miss cycles = I x 36% x 4% x 100 = 1.44 x I
   Total # of memory-stall cycles = 3.44 I
   CPI with memory stalls (it is 2 without) = 3.44 + 2 = 5.44
   Since clock rate and the # of instructions are the same:
   CPU time with stalls/ CPU time with perfect cache =
   I x CPI (stall) x clock cycle time / I x CPI (perfect) x clock cycle time
   = 5.44/2 = 2.72
   → the performance with perfect cache is 2.72 x better

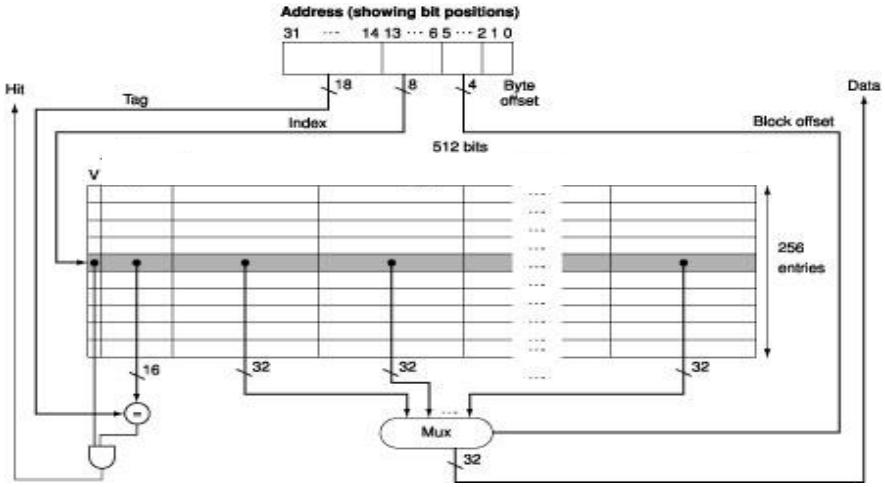
---

2. AMAT = Hit time + Miss rate ×Miss penalty
   AMAT = 2 ns + 0.05 x (20x2ns) = 4 ns

3. AMAT = (1.2x2 ns) + (20×2 ns×0.03) = 2.4 ns + 1.2 ns = 3.6 ns. Yes, it's a good choice

4. (a) Compute the size of the cache shown below. (b) Compute the total number of bits required to implement this cache for the Intrisity FastMATH embedded fast microprocessor (this number represents the total amount of memory needed for storing all data, tags, and valid bits.

(a) Total bits = 256 x 512 = 131,072 bits
(b) Total bits = 256 x (Data + Tag + Valid) = 256 x (512 +18 +1) = 135,936 bits



18

5. Describe the general characteristics of a program that would exhibit very little temporal and spatial locality with regards to data accesses → accessing variables only once and no loops or reuse of instructions

6. Describe the general characteristics of a program that would exhibit very high amounts of temporal locality but very little spatial locality with regards to data accesses. → High temporal locality for code means lots tight loops (in Assembly these loops contains few instructions and iterate many times) and low temporal locality for data accesses means data is scattered: A program that repeatedly branches between few locations spaced far apart in memory.

7. Describe the general characteristics of a program that would exhibit very little temporal locality but very high amounts of spatial locality with regards to data accesses → low temporal locality for code means no loops or branches but very high amounts of spatial locality with regards to data accesses means marching through arrays.

- Low temporal locality for code means no loops and no reuse of instructions.
- High temporal locality for code means tight loops with lots of reuse.
- Low spatial locality for code means lots of jumps to far away places.
- High spatial locality for code means no branches/jumps at all.

8. Consider a memory hierarchy using one of the three organizations for main memory shown below. Assume that the cache block size is 16 words, that the width of organization (b) of the figure is four words, and that the number of banks in organization (c) of the figure is four. If the main memory latency for a new access is 10 memory bus clock cycles and the transfer time is 1 memory bus clock cycle, what are the miss penalties for each of these organizations? Show work for full credit. (3 points)

Configuration (a) requires 16 main memory accesses to retrieve a cache block, ad words of a block are transferred 1 at a time.

Miss penalty = 1 + (16 x 10) + (16 x 1) = 177 clock cycles

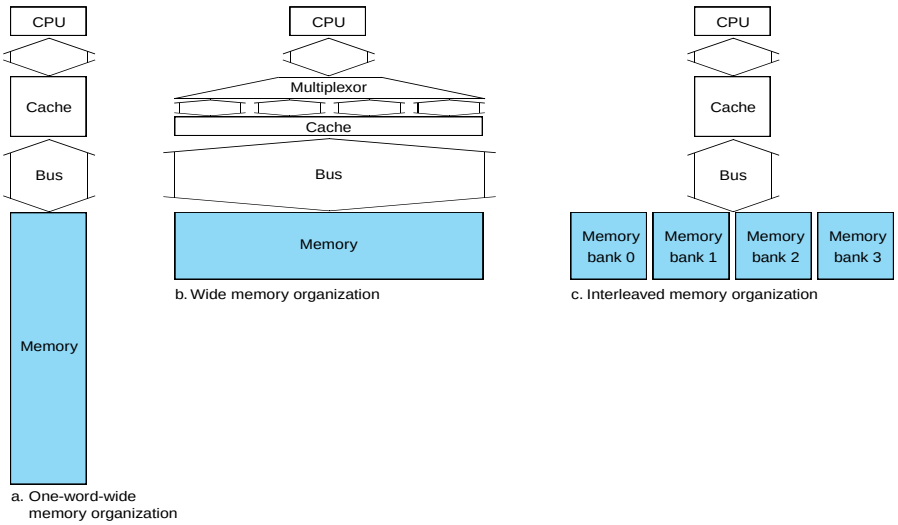
Configuration (b) requires 4 main memory accesses to retrieve a cache block and words of the block are transferred 4 at a time.

Miss penalty = 1 + (16/4 x 10) + (16/4 x 1) = 45 clock cycles

Configuration (c) requires 4 main memory accesses to retrieve a cache block, and words of the block are transferred 1 at a time.

Miss penalty = 1 + (16/4 x 10) + (16 x 1) = 57 clock cycles

Configuration (b) is most favorable option.



Miss rate is the fraction of memory accesses not found in a level of memory hierarchy.
Hit Time is the time required to access a level of memory hierarchy, including the time to determine whether the access is a hot or a miss.
Miss penalty is time required to fetch a block into a level of memory hierarchy (includes time to access the block + time to transfer it from one level to the other.)

- Be able to debug a small MIPS program similar to MIPS 1 OR MIPS 2
  Errors are not syntax related.

Useful Equations

CPU Time_prog = Clock Cycles_prog / Clock Rate
CPU Time_prog = Clock Cycles_prog x Cycle Time
CPI_prog = Clock Cycles_prog / Instructions_prog
CPU Time prog = Instruction Count prog x CPI x Cycle Time
CPU Time prog = (Instruction Count prog x CPI) / clock rate
Average memory access time = Time for a hit + (Miss rate x Miss penalty)
MIPS = Instruction count/(Execution time x 10^6)