1. (a) A benchmark program P is executed to analyze two computer systems S1 and S2. On system S1, each register type (R-type) instruction takes 4 clock cycles, a load instruction takes 5 clock cycles, and a branch instruction takes 3 clock cycles, respectively. On system S2, each R-type instruction takes 3 clock cycles, a load instruction takes 5 clock cycles, and a branch instruction takes 2 clock cycles, respectively. In both systems, the program P executed 80 million R-type instructions, 40 million load instructions, and 25 million branch instructions.

(i) Find the CPI for system S1 and S2.

(3 marks)

Answer

S1 R-type = 4 clock cycles
S1 Load = 5 clock cycles
S1 Branch = 3 clock cycles

S2 R-Type = 3 clock cycles
S2 Load = 5 clock cycles
S2 Branch = 2 clock cycles

P1 R-Type = 80 million
P1 Load = 40 million
P1 Branch = 25 million
P1 Total instructions = 80 + 40 + 25 = 145×10^6

S1 Total clock cycles = (4*80 + 5*40 + 3*25) = 595×10^6
S2 Total clock cycles = (3*80 + 5*40 + 2*25) = 490×10^6

$$\text{S1 CPI} = \frac{\text{S1 Total clock cycles}}{\text{P1 Total instructions}} = \frac{595 \times 10^6}{145 \times 10^6} = \frac{119}{29} = 4.103$$

$$\text{S2 CPI} = \frac{\text{S2 Total clock cycles}}{\text{P1 Total instructions}} = \frac{490 \times 10^6}{145 \times 10^6} = \frac{98}{29} = 3.379$$

1. (a) (ii) Assuming that system S2's clock speed is 15% slower than system S1's clock speed, which system is faster in running program P, and what is the speedup?

(6 marks)

Answer

$$Tc = \frac{1}{Freq}$$

$$\text{Execution time } T = IC * CPI * Tc = \frac{IC * CPI}{Freq}$$

$$\text{S2's clock speed} = (1 - 0.15) * Freq = 0.85(Freq)$$

$$T_{S1} = \frac{IC * 4.103}{Freq}$$

$$T_{S2} = \frac{IC * 3.379}{0.85(Freq)}$$

$$\text{Speedup} = \frac{T_{original}}{T_{enhanced}} = \frac{T_{S1}}{T_{S2}} = \frac{IC * 4.103}{Freq} * \frac{0.85(Freq)}{IC * 3.379} = 1.032$$

1. (b) If a fraction P of a program is enhanced by a factor of F when executed on an enhanced machine, derive the expression of the speedup over the original machine. Based on that expression, find the speedup value when P=0.4 and F=1000.

(4 marks)

Answer

P = 0.4

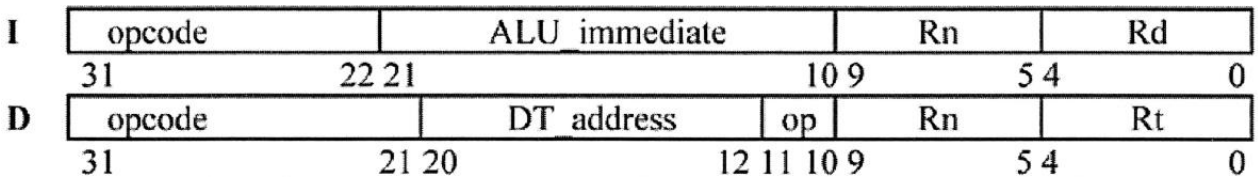
F = 1000

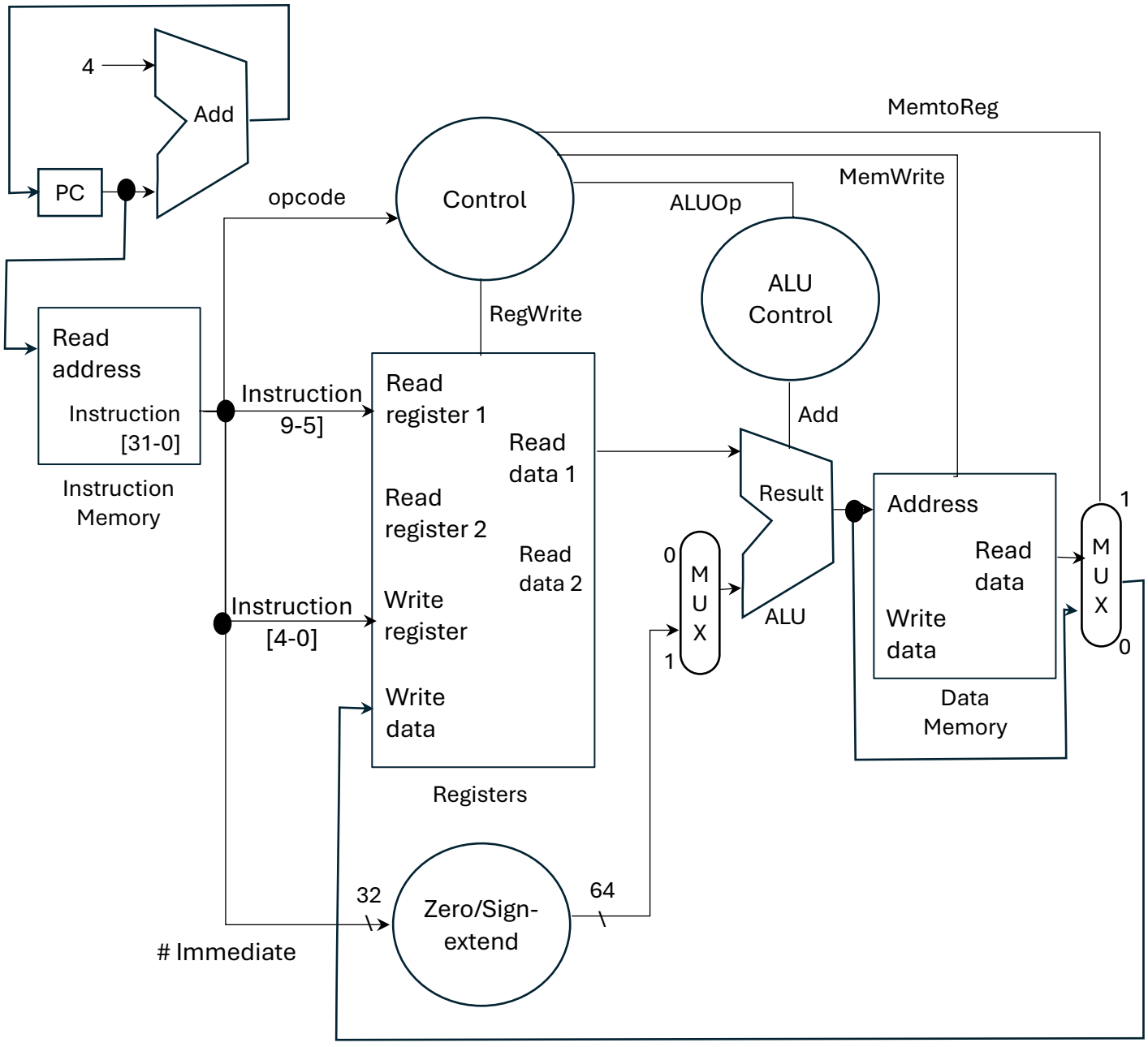
$$\text{Speedup (P,F)} = \frac{1}{(1 - P) + \frac{P}{S}} = \frac{1}{(1 - 0.4) + \frac{0.4}{1000}} = \frac{2500}{1501} = 1.67$$

1. (c) Use a neat diagram to show the datapath of a single-cycle architecture that supports the execution of both "LDUR X2, [X1, #8]" and "ADDI X4, X3, #5". The datapath needs to have minimal number of multiplexers (control signals can be simplified). Briefly explain the working of the instructions "LDUR X2, [X1, #8]" and "ADDI X4, X3, #5".

(12 marks)

Answer





Working of the Instructions

"LDUR X2, [X1, #8]"

1. Read the data from register X2.
2. Compute the memory address by adding the values of register X2 to the immediate value “8” using the ALU.
3. Write the values from the computed memory address to the register X0.
4. [X2] ← mem[[X1] + [8]]

"ADDI X4, X3, #5"

1. Read the data from register X3.
2. Compute the result by adding the values of register X3 to the immediate value “5” using the ALU.
3. Write the computed result to the register X4.
4. [X4] ← [X3] + [5]

2. (a) Listing Q2 shows a code segment that is intended to be executed in a 5-stage pipelined LEGv8 processor. The program counter is updated with the branch target address at the Execute stage. Let the initial values in hexadecimal be X4=0x0000000000010000 and X5=0x0000000000001000 (CBNZ: *branch if not equal to 0*).

Listing Q2

```
I1   loop: LDUR  X0,  [X4,  #0]
I2         LDUR  X1,  [X5,  #0]
I3         SUB   X2,  X1,   X0
I4         STUR  X2,  [X4,  #0]
I5         SUBI  X4,  X4,   #16
I6         SUBI  X5,  X5,   #16
I7         CBNZ  X5,  loop
```

(i) Calculate the steady-state CPI of the code segment in Listing Q2 with the help of a reservation table for the execution of the code if full data forwarding is allowed. Show the forwarded paths and the dependencies. Find the total number of loop iterations.

(9 marks)

(ii) The code segment shown in Listing Q2 is now intended to be executed in a two-way superscalar processor. In the superscalar processor, one way is exclusively for load and store instructions, whereas the other way can execute all instructions except load and store. Find the CPI achieved by the superscalar architecture. Note that full data forwarding is allowed.

(7 marks)

(iii) Briefly explain the concept of delayed branching with an example. How is this technique used to improve the execution time of a processor?

(5 marks)

2 (a) (i) Answer

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
loop: LDUR X0, [X4, #0]	I1.	F	D	E	M	W										
LDUR X1, [X5, #0]	I2.		F	D	E	M	W									
SUB X2, X1, X0	I3.			F	D	S	E	M	W							
STUR X2, [X4, #0]	I4.				F		D	E	M	W						
SUBI X4, X4, #16	I5.						F	D	E	M	W					
SUBI X5, X5, #8	I6.							F	D	E	M	W				
CBNZ X5, loop	I7.								F	D	E	M	W			
										S	S	F	D	E	M	W

Data dependency:
RAW: I1, I3 at X0
I2, I3 at X1
I3, I4 at X2
I6, I7 at X5

WAR: I1, I5 at X4
I4, I5 at X4
I6, I7 at X5

$$\text{Steady state CPI} = \frac{\text{No. of instructions + No. of stalls + No. of control}}{\text{No. of instructions}}$$

$$= \frac{7 + 1 + 2}{7} = \frac{10}{7} = 1.43$$

X5 = 0x0000000000001000 (hex)
= 4096 (ten)

$$\text{Total number of loop iterations} = \frac{4096}{8} = 512$$

2 (a) (ii) Answer

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
loop: LDUR X0, [X4, #0]	I1.	F	D	E	M	W										
LDUR X1, [X5, #0]	I2.		F	D	E	M	W									
SUB X2, X1, X0	I3.			F	D	S	E	M	W							
STUR X2, [X4, #0]	I4.				F		D	E	M	W						
SUBI X4, X4, #16	I5.						F	D	E	M	W					
SUBI X5, X5, #8	I6.							F	D	E	M	W				
CBNZ X5, loop	I7.								F	D	E	M	W			
										S	S	F	D	E	M	W

	Way-1 (rest of instructions)	Way -2 (LDUR and STUR only)	Cycle
loop	nop	LDUR X0, [X4, #0]	1
	nop	LDUR X1, [X5, #0]	2
	SUBI X5, X5, #8	nop	3
	SUB X2, X1, X0	nop	4
	CBNZ X5, loop	STUR X2, [X4, #0]	5
	SUBI X4, X4, #16	nop	6
	nop	nop	7

2-way super-scalar after instruction reordering

$$CPI = \frac{7}{7}$$

$$= 1$$

2 (a) (iii) Answer

Combined effect of superscalar architecture and instruction reordering helped us improve the performance of the system.

2 (b) Briefly explain the concept of delayed branching with an example. How is this technique used to improve the execution time of a processor?

(5 marks)

Answer

Delayed branching (aka Static Prediction) is a technique used in pipelined processors to minimize the performance penalty caused by branch instructions.

When a branch instruction is encountered, the system has to compute whether branch taken or not taken.

During this time, the pipeline may stall because the next instruction cannot be fetched until the branch outcome is known.

To avoid pipeline stalls, delayed branching fills the delay slot(s), the instruction cycle(s) immediately following the branch, with an instruction that can be executed regardless of the branch outcome.

Example

```
LDUR X1, [X5, #0]                LDUR X1, [X5, #0]
SUB  X2, X1, X0          ⇨       SUBI X7, X8, #8
                                 SUB  X2, X1, X0
```

3. (a) Consider a Byte-addressable memory with the address space of 32 bits. A 64 KB (1 KB = 1024 Bytes) eight-way set-associative cache is used for this memory system with the cache block size of 256 Bytes. Determine the number of bits for the tag, set index and block offset fields of the address, respectively.

(8 marks)

Answer

Address size = 32 bits
Cache size = 64 KB
No. of way = 8
Block size = 256 Bytes

$$\text{No. of block} = \frac{\text{Cache size}}{\text{Block size}} = \frac{64 * 1024}{256} = 256 \text{ Bytes}$$

$$\text{No. of set} = \frac{\text{No. of block}}{\text{No. of way}} = \frac{256}{8} = 32$$

Offset = $\log_2$(Block size) = $\log_2(256)$ = 8 bits
Index = $\log_2$(No. of set) = $\log_2(32)$ = 5 bits

Address size = Tag + Index + Offset
Tag = Address size – Index – Offset
= 32 - 5 - 8
= 19 bits

Address		
Tag	Index	Offset
19	5	8

3. (b) Tables Q3a and Q3b show the miss rates and average memory access times of a cache system configured in different cache and block sizes.

Table Q3a: Actual Miss Rate v.s. Block Size

Block Size (Bytes)	Cache Size			
	4KB	16KB	64KB	256KB
64	7.00%	2.64%	X%	0.51%
256	9.51%	3.29%	1.15%	0.49%

Table Q3b: Average Memory Access Time v.s. Block Size

Block Size (Bytes)	Cache Size			
	4KB	16KB	64KB	256KB
64	7.16	Y	1.93	Z
256	11.65	4.69	2.29	1.55

3. (i) Assume that the hit time takes 1 clock cycle, and the memory system takes 80 clock cycles of overhead and then delivers 16 bytes every 2 clock cycles. For example, it can supply 16 bytes in 82 clock cycles, 32 bytes in 84 clock cycles, 64 bytes in 88 clock cycles, and so on. Compute the missing entries X, Y and Z in Tables Q3a and Q3b.

(8 marks)

Answer

AMAT = Hit time + Miss rate * Miss penalty

1.93 = 1 + X% * 88 ⇒ X = 1.05%
Y = 1 + 2.64% * 88 ⇒ Y = 3.32
Z = 1 + 0.51% * 88 ⇒ Z = 1.45

(ii) Between cache miss rate and average memory access time, which one is the more important performance to the cache performance? Briefly justify your answer.

(3 marks)

Answer

Average memory access time (AMAT) is the more important performance to the cache performance because it considers both the time spent on cache hits and misses.

3. (c) Name two cache replacement algorithms and two cache write policies for cache system management. For the two cache write policies, which one has better performance considering the huge difference in speed between the cache and the main memory in a computer?

(6 marks)

Answer

Two basic cache write policies

- Write-through policy
- Write-back policy

Write-back policy has better performance

- CPU writes occur at the speed of the cache memory
- multiple writes within a block require only one write to the lower-level memory
- write back uses less memory bandwidth and dissipate less power

4 (a) Briefly explain why the flow control of a conditional program is managed differently on GPUs and CPUs.

(4 marks)

Answer

CPUs are designed for efficient sequential execution and precise control flow, while GPUs are optimized for parallel execution and efficient handling of large data sets.

The different approaches to flow control reflect these fundamental architectural differences.

(b) Answer the following questions based on the CUDA code in Figure Q4a.

(4 marks)

```
Line
1  __global__ compute (float *a, float *b, int BLOCKSIZE) {
2    __shared__ s_a[128], s_b[128];
3    /* copy portion of input data into shared memory */
4    s_a[threadIdx. x] = a[blockIdx.x*BLOCKSIZE+threadIdx.x];
5
6    /* Time step loop */
7    for (int t = 0; t<MAX_TIME; t++) {
8       /*alternate inputs and outputs on even/odd time steps*/
9       if (t % 2 == 0) {
10        int boundary = min((blockIdx.x+1) *BLOCKSIZE-1,
11                           blockDim. x*BLOCKSIZE-1, threadIdx+2);
12        s_b[threadIdx. x] = s_a[threadIdx.x] + s_a[boundary];
13      }
14
15      else /* (t%2 == 1) */ {
16        int boundary = min((blockIdx.x+1) *BLOCKSIZE-1,
17                          blockDim. x*BLOCKSIZE-1, threadIdx+2);
18        s_a[threadIdx.x] = s_b[threadIdx. x] + s_b[boundary];
19      }
20   }
21
22   /* Result is in s b, and must be copied to b */
23   b[blockIdx.x*BLOCKSIZE+threadIdx.x] = s_b[threadIdx.x];
24 }
```

Figure Q4a

4. (b) (i) Add synchronization to the code in Figure Q4a to derive a correct implementation that has no race conditions. (Hint: You should be able to simply insert `__synchthreads()` calls without modifying the code.)

(6 marks)

Answer

Line 5: `__syncthreads();`

Line 19: `} __syncthreads();`

(ii) What is the number of threads in a warp in a typical Nvidia GPU design? Assume a Stream Multiprocessor (SM) in a GPU has sufficient register and shared memory resources to reside all the blocks. What is the total number of warps that will be created by launching the kernel as compute << 8,512>>?

(5 marks)

Answer

No. of threads in a warp = 32

No. of threads per block = 512

No. of warp per block $= \frac{512}{32} = 16$

No. of block = 8

Total no. of warps = No. of warp per block * No. of block
= 16 * 8
= 128 warps

(iii) Briefly explain functions of the `__global__` and `__shared__` specifiers in CUDA programming.

(4 marks)

Answer

- `__global__` is used to declare a kernel function as device code
  - Kernel function is called from the host

- `__shared__` is used to declare a variable/array in shared memory
  - So that data is shared between threads in a block

4. (c) Consider the following sequence of actual outcomes of a branch instruction (N, N, N, N, N, N, N, T, N, T, N, T, N), where T means that the branch is taken, and N means that the branch is not taken. Assume that there is only one branch instruction in the program. What is the prediction accuracy for the last 6 occurrences of this branch, i.e., (T, N, T, N, T, N), if the 2-bit branch predictor as shown in Figure Q4b is applied?

(6 marks)

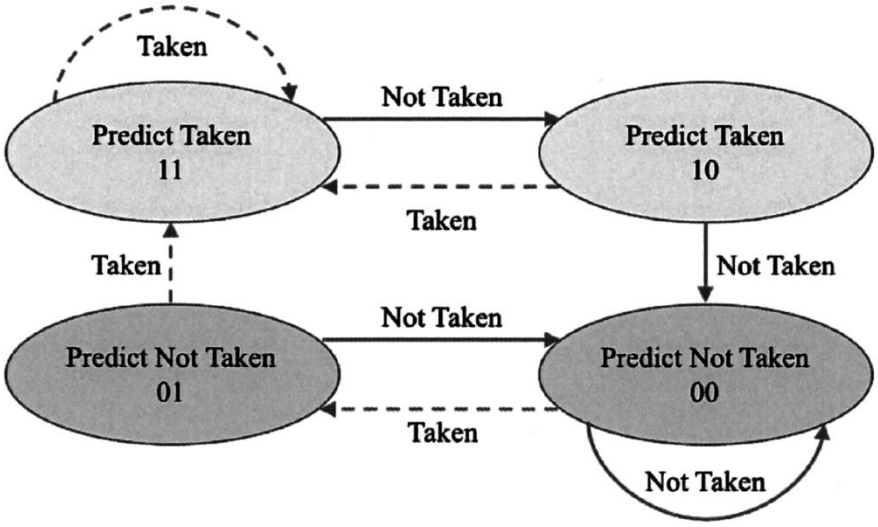


Figure Q4b

Answer

After the first 7 outcomes are not taken (N), we can assume that the state will be at "Predict Not Taken 00".

Predictor State	00	01	00	01	00	01
Prediction Decision	N	N	N	N	N	N
Actual Outcome	T	N	T	N	T	N
Correct Prediction? (Y/N)	N	Y	N	Y	N	Y

Prediction accuracy = 3 / 6 = 50%

Appendix – Instruction Formats