1. (a) Machine M1 runs at its maximum operating clock frequency of 300 MHz at operating voltage of 5 V. If the operating voltage of the processor is reduced to 3 V, find the percentage change in the dynamic power consumption and static power consumption, provided that the processor runs at its maximum possible clock frequency at 3 V.

(5 marks)

Answer

Dynamic power consumption

The maximum operating clock frequency is proportional to operating voltage.

$P1_{dyn} = A * C * 5 * 5 * f_1$

$P2_{dyn} = A * C * 3 * 3 * f_2$

$P_{dyn} = ACV^2f$

$\frac{f_2}{f_1} = \frac{3}{5} = 0.6 \quad \Rightarrow \quad f_2 = 0.6f_1$

$\frac{P2_{dyn}}{P1_{dyn}} = \frac{3 * 3 * 0.6f_1}{5 * 5 * f_1} = \frac{27}{125} = 0.216$

⇒ 78.4% decrease in dynamic power consumption

Static power consumption

$P1_{st} = 5 * I_{leak}$

$P2_{st} = 3 * I_{leak}$

$P_{st} = VI_{leak}$

$\frac{P2_{st}}{P1_{st}} = \frac{3}{5} = 0.6$

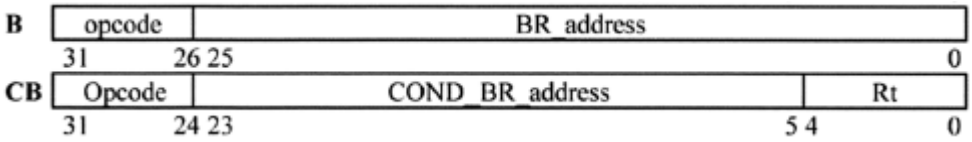
⇒ 40% decrease in static power consumption

1. (b) The hexadecimal value of the content of the program counter (PC) in a LEGv8 processor is 0x500000AC as shown in Table Q1b. The code needs to move to 0x50F000A8. Identify the branch instruction (either conditional or unconditional branch) that needs to be used and the required offset in hexadecimal. Justify your answer.

(5 marks)

Table Q1b

Address	Instruction
0x500000A8	ADD X1, X31, X31
0x500000AC	Conditional / Unconditional Branch to loop
.....	.....
0x50F000A8 (loop)	LDUR X5, [X2, #0]



Answer

Offset = (0x50F000A8 – 0x500000AC) / 4
= 0x00EFFFFC / 4
= 0x003BFFFF

No. of address bits for unconditional branch = 25 – 0 + 1 = 26 bits

The maximum positive number for 26 bits
= 01 1111 1111 1111 1111 1111 1111 (0x1FFFFFF)

No. of address bits for conditional branch = 23 – 5 + 1 = 19 bits

The maximum positive number for 19 bits
= 011 1111 1111 1111 1111 (0x3FFFF)

As offset values 0x003BFFFF has exceeded the maximum offset of conditional branch 0x0003FFFF, hence only unconditional branch can be applied here.

Unconditional Branch instructions would be
= B offset
= B, 0x003BFFFF

1. (c) Find the largest address to which the LEGv8 code can branch based on the branch instruction identified in Q1(b).

(3 marks)

Answer

Maximum PC = PC + Sign extended (offset) << 2
= 0x500000AC + (0x1FFFFFF)*4
= 0x5800000A8

1. (d) Figure Q1 shows a single-cycle LEGv8 datapath. The propagation delays for the major functional components are listed in Table Q1c. PC++ indicates the delay for PC increment. I-mem (R) indicates the time for reading instruction memory and D-mem (R/W) indicates the time required for read or write operation on data memory. The delays of shifter, sign-extend and AND gates are assumed to be negligibly small.

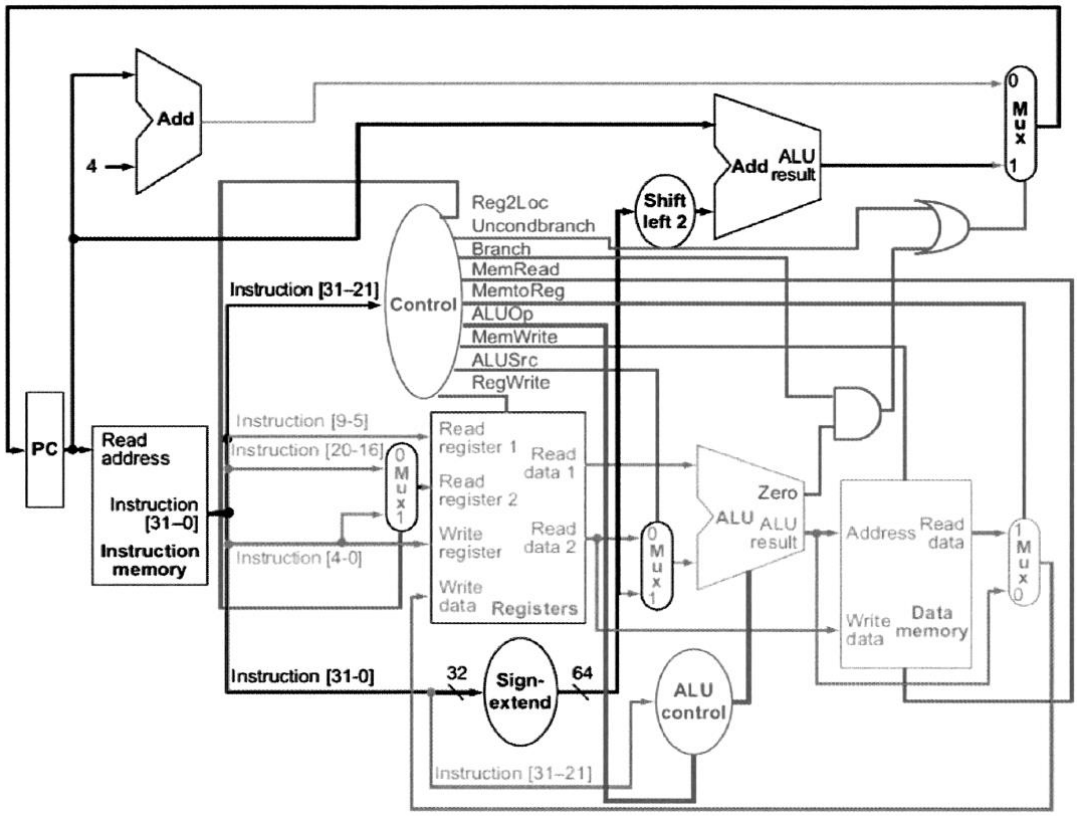


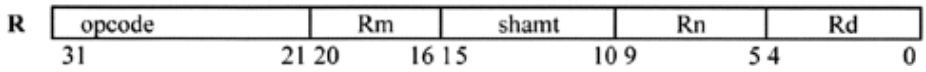
Figure Q1

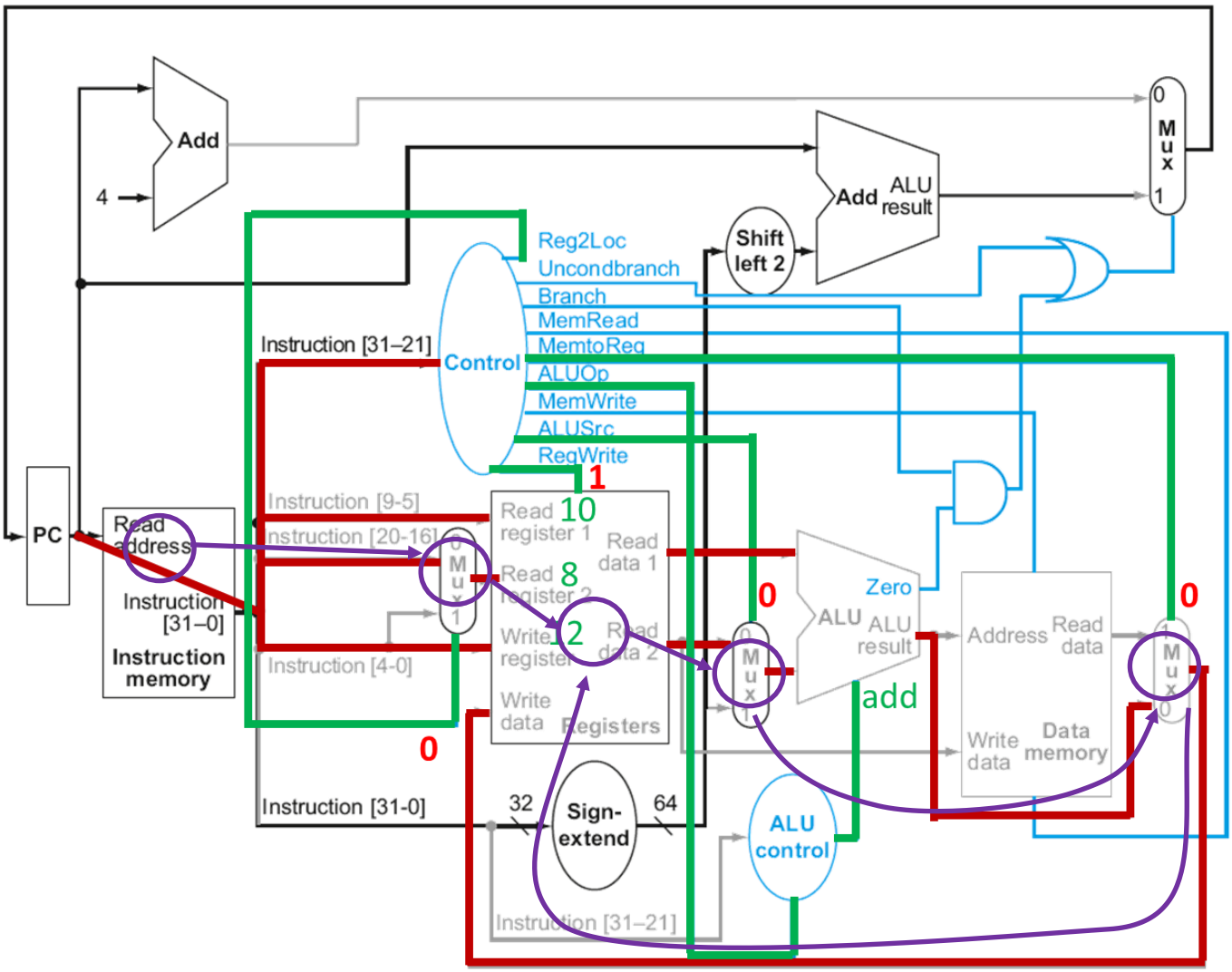
Table Q1c

PCin→ PCout	PC++	Adder	Mux	ALU	Register (R/W)	Control logic	D-mem (R/W)	I-mem (R)
100ps	250ps	200ps	20ps	250ps	200ps	200ps	500ps	500ps

1. (d) (i) What is the minimum clock period for ADD, STUR, CBZ and B instructions? (8 marks)

Answer



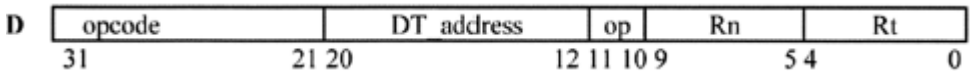


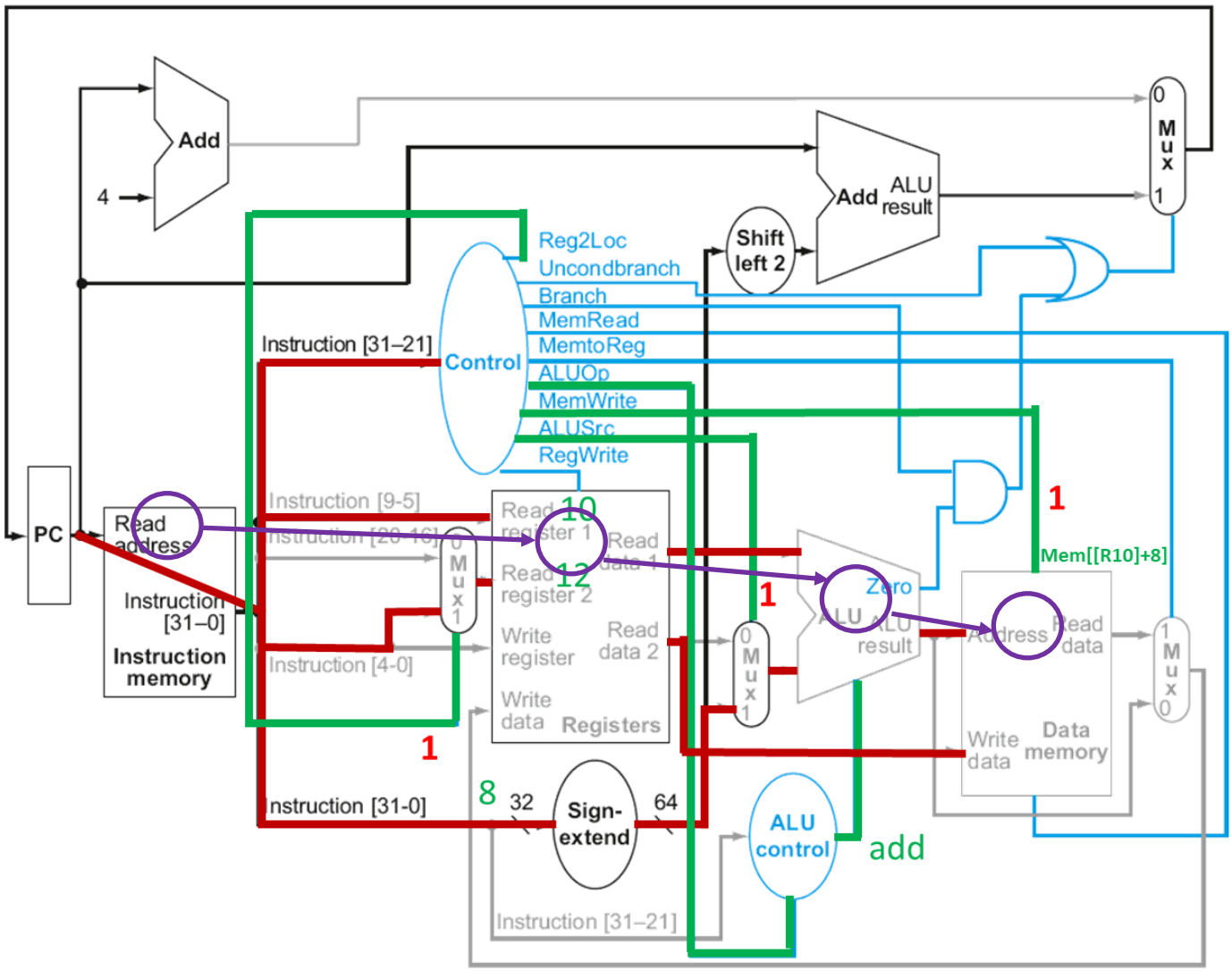
ADD instruction latency

I-mem + mux(before regfile) + Reg(R) + mux(before ALU) + ALU + mux(WB) + Reg(W)

= 500 + 20 + 200 + 20 + 250 + 20 + 200

= 1210 ps





STUR instruction latency

I-mem + Reg(R) + ALU + D-MEM

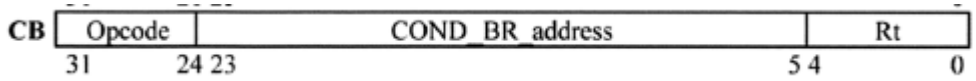
= 500 + 200 + 250 + 500

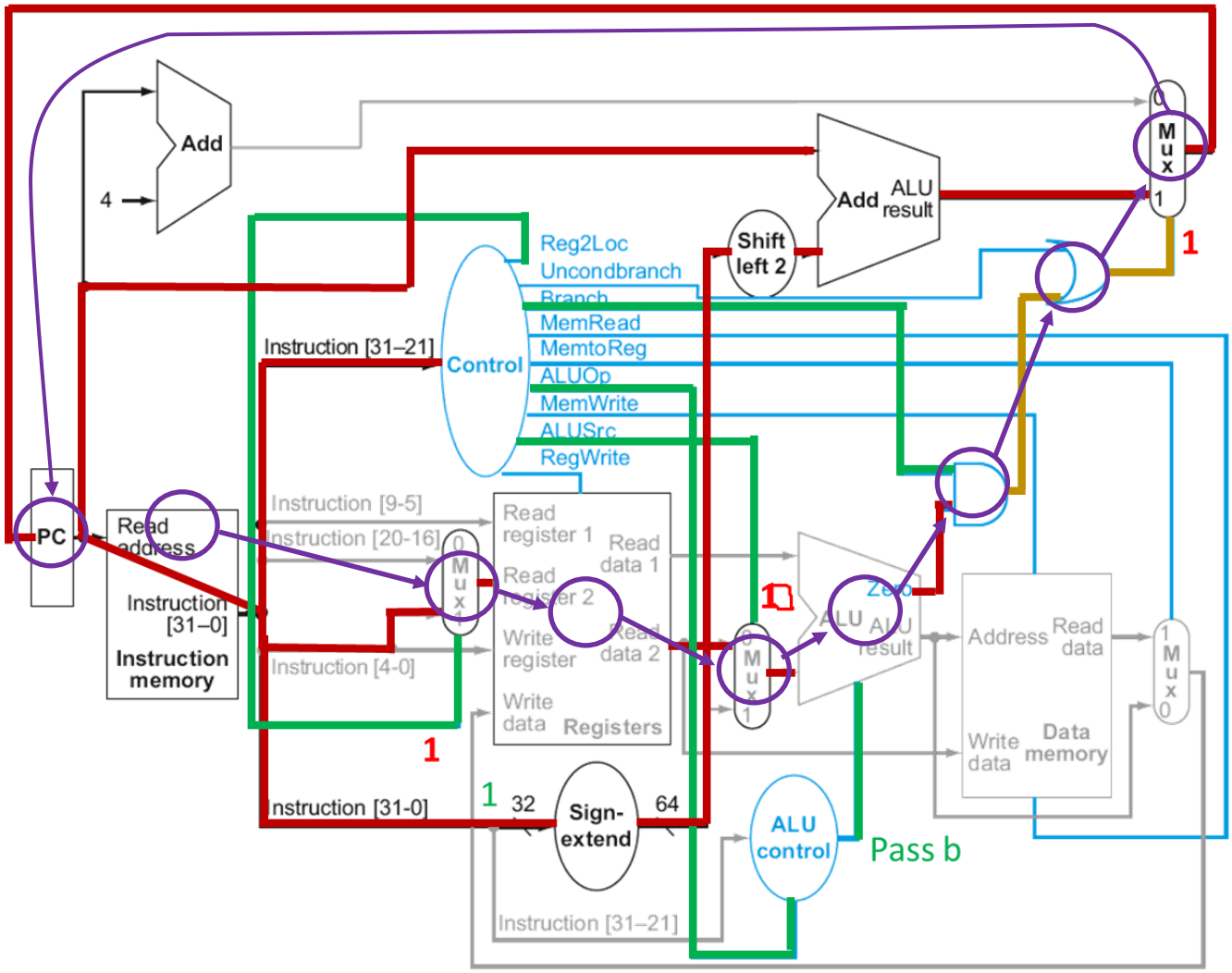
= 1450 ps

500 20

The mux(before regfile) is not part of latency as Latency of (I-mem + mux(before regfile) + D-MEM) < the above given latency

500



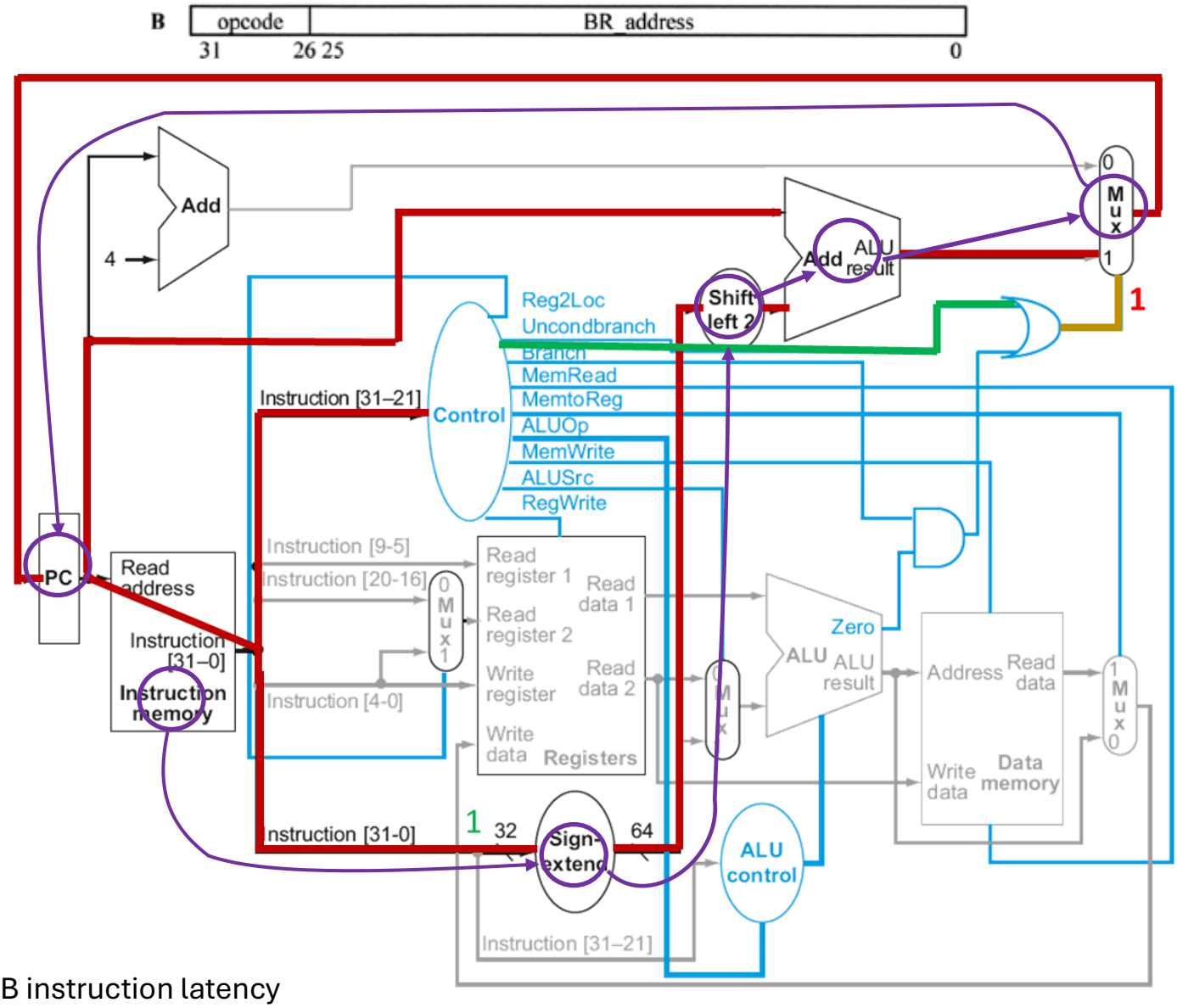


CBZ instruction latency

I-mem + mux(before regfile) + Reg(R) + mux(before ALU) + ALU + AND gate + OR gate + mux(branch) + PCin→PCout

= 500 + 20 + 200 + 20 + 250 + 0 + 0 + 20 + 100

= 1110 ps



B instruction latency

I-mem + Sign-ext + Shift-Left-2 + Adder + mux(branch) + PCin→PCout

= 500 + 0 + 0 + 200 + 20 + 100

= 820 ps

The minimum clock frequency:

- ADD instruction = 1210 ps ⇒ R-format
- STUR instruction = 1450 ps ⇒ D-format
- CBZ instruction = 1110 ps ⇒ CB-format
- B instruction = 820 ps ⇒ B-format

1 (d) (ii) What is the maximum operation frequency assuming that the machine operates only R, D and B format instructions?

(4 marks)

Answer

- ADD instruction = 1210 ps ⇒ R-format
- STUR instruction = 1450 ps ⇒ D-format
- B instruction = 820 ps ⇒ B-format

LDUR instruction latency (D-format)
= I-mem + Reg(R) + ALU + D-mem + mux(W) + Reg(W)
= 500 + 200 + 250 + 500 + 20 + 200
= 1670 ps

The maximum operation frequency = $\frac{1}{1670 \times 10^{-9}}$ = 598.8 kHz

2. 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 Decode stage. Let the initial values in hexadecimal be X7=0x0000000000010000 and X8=0x0000000000001100. Answer the following questions (CBNZ: *branch if not equal to 0*).

```
I1    loop:  LDUR  X0,  [X7,  #0]
I2           LDUR  X1,  [X8,  #400]
I3           AND   X2,  X1,   X0
I4           XORI  X3,  X2,   0x00F
I5           STUR  X3,  [X7,  #0]
I6           SUBI  X7,  X7,   #8
I7           SUBI  X8,  X8,   #16
I8           CBNZ  X8,  loop
     Finish
```

Listing Q2

(a) 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, assuming full data forwarding is allowed. Show the forwarded paths and the dependencies. Find the total number of loop iterations.

(7 marks)

(b) 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 instruction, 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.

(6 marks)

(c) Perform unrolling by a factor of 2 and do necessary reordering. Use the unrolled and reordered code segment to be executed in a two-way superscalar machine. 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. Comment on the change in CPI when compared to Q2(b).

(10 marks)

(d) Briefly comment on the impact of applying static branch prediction of "Always Not Taken" in Q2(b) and Q2(c).

(2 marks)

2 (a) Answer

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
loop: LDUR X0, [X7, #0]	I1.	F	D	E	M	W										
LDUR X1, [X8, #400]	I2.		F	D	E	M	W									
AND X2, X1, X0	I3.			F	D	S	E	M	W							
XORI X3, X2, 0x00F	I4.				F		D	E	M	W						
STUR X3, [X7, #0]	I5.						F	D	E	M	W					
SUBI X7, X7, #8	I6.							F	D	E	M	W				
SUBI X8, X8, #16	I7.								F	D	E	M	W			
CBZ X8, loop	I8.									F	D	E	M	W		
											S	F	D	E	M	W

Data dependency:
RAW: I1, I2 at X0
I2, I3 at X2
I3, I4 at X2
I4, I5 at X3
I7, I8 at X8

WAR: I1, I6 at X7
I5, I6 at X7
I7, I8 at X8

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

$$= \frac{8 + 1 + 1}{8} = \frac{5}{4} = 1.25$$

X8 = 0x0000000000001100 (hex)
= 4352 (ten)

$$\text{Total number of loop iterations} = \frac{4352}{16} = 272$$

2 (b) Answer

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
loop: LDUR X0, [X7, #0]	I1.	F	D	E	M	W										
LDUR X1, [X8, #400]	I2.		F	D	E	M	W									
AND X2, X1, X0	I3.			F	D	S	E	M	W							
XORI X3, X2, 0x00F	I4.				F		D	E	M	W						
STUR X3, [X7, #0]	I5.						F	D	E	M	W					
SUBI X7, X7, #8	I6.							F	D	E	M	W				
SUBI X8, X8, #16	I7.								F	D	E	M	W			
CBZ X8, loop	I8.									F	D	E	M	W		
											S	F	D	E	M	W

	Way-1 (rest of instructions)	Way -2 (LDUR and STUR only)	Cycle
loop	nop	LDUR X0, [X7, #0]	1
	nop	LDUR X1, [X8, #400]	2
	SUBI X8, X8, #16	nop	3
	AND X2, X1, X0	nop	4
	XORI X3, X2, 0x00F	nop	5
	CBZ X8, loop	STUR X3, [X7, #0]	6
	SUBI X7, X7, #8	nop	7

2-way super-scalar after instruction reordering

$$CPI = \frac{7}{8}$$

$$= 0.875$$

2 (c) Answer

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
loop: LDUR X0, [X7, #0]	I1.	F	D	E	M	W										
LDUR X1, [X8, #400]	I2.		F	D	E	M	W									
AND X2, X1, X0	I3.			F	D	S	E	M	W							
XORI X3, X2, 0x00F	I4.				F		D	E	M	W						
STUR X3, [X7, #0]	I5.						F	D	E	M	W					
SUBI X7, X7, #8	I6.							F	D	E	M	W				
SUBI X8, X8, #16	I7.								F	D	E	M	W			
CBZ X8, loop	I8.									F	D	E	M	W		
											S	F	D	E	M	W

Unrolling by 2

```
loop: LDUR  X0,  [X7,  #0]
      LDUR  X1,  [X8,  #400]
      AND   X2,  X1,  X0
      XORI  X3,  X2,  0x00F
      STUR  X3,  [X7,  #0]
      LDUR  X4,  [X10, #-8]
      LDUR  X5,  [X11, #384]
      AND   X6,  X5,  X4
      XORI  X9,  X6,  0x00F
      STUR  X9,  [X10, #-8]
      SUBI  X7,  X7,  #16
      SUBI  X8,  X8,  #32
      CBNZ  X8,  loop
```

```
I1
I2
I3
I4
I5
I6
I7
I8
I9
I10
I11
I12
I13
```

Unrolling by 2 and reordering

```
loop: LDUR  X0,  [X7,  #0]
      LDUR  X1,  [X8,  #400]
      LDUR  X4,  [X10, #-8]
      LDUR  X5,  [X11, #384]
      AND   X2,  X1,  X0
      AND   X6,  X5,  X4
      XORI  X3,  X2,  0x00F
      XORI  X9,  X6,  0x00F
      STUR  X3,  [X7,  #0]
      STUR  X9,  [X10, #-8]
      SUBI  X7,  X7,  #16
      SUBI  X8,  X8,  #32
      CBNZ  X8,  loop
```

	Way-1 (rest of instructions)	Way -2 (LDUR and STUR only)	Cycle
loop	nop	LDUR X0, [X7, #0]	1
	nop	LDUR X1, [X8, #400]	2
	SUBI X8, X8, #32	LDUR X4, [X10, #-8]	3
	AND X2, X1, X0	LDUR X5, [X11, #384]	4
	XORI X3, X2, 0x00F	nop	5
	AND X6, X5, X4	STUR X3, [X7, #0]	6
	XORI X9, X6, 0x00F	nop	7
	CBNZ X8, loop	STUR X9, [X10, #-8]	8
	SUBI X7, X7, #16	nop	9

2-way super-scalar after 2-unrolling and instruction reordering

$$CPI = \frac{9}{13}$$

$$= 0.642$$

Combined effect of loop unrolling and instruction reordering helped us improve the performance of the system.

2 (d) Answer

- Execute successive instructions in sequence.
- For the conditional branch, if the branch is not taken, there will be no stall.
- Flush the pipeline and read correct instructions if branch actually taken.

3. (a) Name three cache organization schemes.

(3 marks)

Answer

- Direct-mapped cache
- Fully associative cache
- Set associative cache

3. (b) Consider a tiny memory system composed of a Byte-addressable main memory and a two-way set-associative cache. The memory size is 4 KB (1 KB = 1024 Bytes). The cache size is 512 Bytes. The block size of 16 Bytes is shared between the main memory and the cache. Determine the number of bits for the tag, set index and block offset fields of this cache system, respectively.

(6 marks)

Answer

No. of way = 2
Memory size = 4 KB
Cache size = 512 Bytes
Block size = 16 Bytes

$$\text{No. of block} = \frac{\text{Cache size}}{\text{Block size}} = \frac{512}{16} = 32$$

$$\text{No. of set} = \frac{\text{No. of block}}{\text{No. of way}} = \frac{32}{2} = 16$$

Address size = $\log_2$(Memory size) = $\log_2(4 * 1024)$ = 12 bits
Offset = $\log_2$(Block size) = $\log_2(16)$ = 4 bits
Index = $\log_2$(No. of set) = $\log_2(16)$ = 4 bits
Address size = Tag + Index + Offset
Tag = Address size – Index – Offset = 12 – 4 – 4 = 4 bits

3. (c) Assume that the cache system as in Q3(b) is initially empty. The following sequence of memory load accesses is conducted: "0x003, 0x0c4, 0x101, 0x002, 0x406, 0x2c2, 0x00c, 0x2c0". The least recently used algorithm is used to handle cache replacement during the execution. Find the hit rate of the cache. Complete Table Q3 below to indicate the hit/miss status for each access.

Table Q3

Time	0	1	2	3	4	5	6	7
Access	0x003	0x0c4	0x101	0x002	0x406	0x2c2	0x00c	0x2c0
Tag								
Index								
Offset								
Hit/Miss?								

(8 marks)

Answer

Address = Tag + Index + Offset = 4 + 4 + 4 bits

Time	Access (hex)	Access (Tag, Index, Offset)
0	0x003	0000 0000 0011
1	0x0c4	0000 1100 0100
2	0x101	0001 0000 0001
3	0x002	0000 0000 0010
4	0x406	0100 0000 0110
5	0x2c2	0010 1100 0010
6	0x00c	0000 0000 1100
7	0x2c0	0010 1100 0000

Time	0	1	2	3	4	5	6	7
Access	0x003	0x0c4	0x101	0x002	0x406	0x2c2	0x00c	0x2c0
Tag	0000	0000	0001	0000	0100	0010	0000	0010
Index	0000	1100	0000	0000	0000	1100	0000	1100
Offset	0011	0100	0001	0010	0110	0010	1100	0000
Hit/Miss?								

3 (c) Answer (continue)

No. of way = 2

Time	0	1	2	3	4	5	6	7
Access	0x003	0x0c4	0x101	0x002	0x406	0x2c2	0x00c	0x2c0
Tag	0000	0000	0001	0000	0100	0010	0000	0010
Index (Set)	0000	1100	0000	0000	0000	1100	0000	1100
Offset	0011	0100	0001	0010	0110	0010	1100	0000
Hit/Miss?	M	M	M	H	M	M	H	H

Access		Way 1		Way 2		Hit/Miss
Initial	SET	TAG	LRU	TAG	LRU	
	0000					
	1100					
0	0000	0000	0			M
	1100					
1	0000	0000	0			
	1100	0000	0			M
2	0000	0000	1	0001	0	M
	1100	0000	0			
3	0000	0000	0	0001	1	H
	1100	0000	0			
4	0000	0000	1	0100	0	M
	1100	0000	0			
5	0000	0000	1	0100	0	
	1100	0000	1	0010	0	M
6	0000	0000	0	0100	1	H
	1100	0000	1	0010	0	
7	0000	0000	0	0100	1	
	1100	0000	1	0010	0	H

Hit rate = 3 / 8 = 0.375 = 37.5%

3. (d) Suppose the latency to access the cache is 1 cycle, and the latency to access the main memory is 80 cycles. Given the cache hit rate as in Q3(c) above, calculate the average memory access time of the memory system.

(4 marks)

Answer

Hit rate = 37.5%
Hit time = 1
Main memory = Miss penalty = 80 cycles

Miss rate = 1 – Hit rate = 1 – 37.5% = 62.5%

AMAT = Hit time + Miss rate * Miss penalty
= 1 + 62.5% * 80
= 51 cycles

3. (e) Based on the memory system in Q3(d), we add a new L2 cache to further enhance the memory access performance. The latency of the L2 cache access is 4 cycles, and the hit rate of the L2 cache is 95%. What is the average memory access time of the new memory system?

(4 marks)

Answer

L2 Cache = L2 Hit time = 4 cycles
L2 Hit rate = 95%

L2 Miss rate = 1 – L2 Hit time = 1 – 95% = 5%
L2 AMAT = 4 + 5% * 80 = 8 cycles

L1 + L2 AMAT = L1 Hit time + L1 Miss rate * L2 AMAT
= 1 + 62.5% * 8
= 6 cycles

4. (a) Assume a branch predictor uses a two-bit prediction scheme as shown in Figure Q4a. The predictor is initialized to the state of "10". Consider a branch instruction with the following repeating sequence of actual outcome: 'N N T N T N", where T indicates the branch is taken and N indicates the branch is not taken.

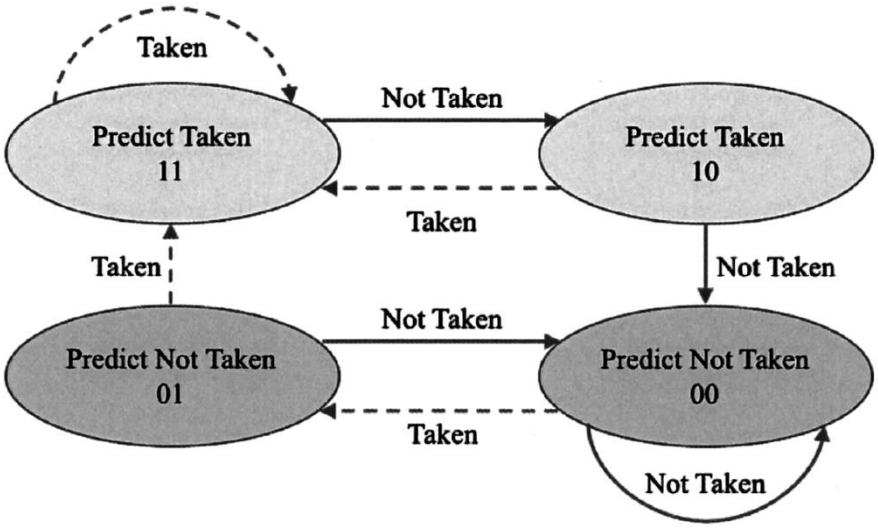


Figure Q4a

(i) Find the prediction accuracy of the two-bit predictor if the repeating sequence of actual outcome is repeated infinitely. Complete Table Q4 below to indicate the prediction decision in each step.

Table Q4

Predictor State	10											
Prediction Decision	T											
Actual Outcome	N	N	T	N	T	N	N	N	T	N	T	N
Correct Prediction?												

(6 marks)

4. (a) (i) Answer

Predictor State	10	00	00	01	00	01	00	00	00	01	00	00
Prediction Decision	T	N	N	N	N	N	N	N	N	N	N	N
Actual Outcome	N	N	T	N	T	N	N	N	T	N	T	N
Correct Prediction?	N	Y	N	Y	N	Y	Y	Y	N	Y	N	Y

Prediction accuracy = 7 / 12 = 58.33%

4 (a) (ii) Compare the prediction accuracy with a static "Always Taken" predictor. Briefly comment on the better choice of predictor in this case using no more than 2 sentences.

(3 marks)

Answer

Prediction for "Always Not Taken" accuracy (all "T" correct predictions)
= 8 / 12 = 66.67%

The "Always Not Taken" predictor has better accuracy in this case because the majority of outcomes are not taken (N).

4. (b) Explain how the following two functions are used in CUDA programs in no more than 2 sentences each.

- `syncthreads();`
- `cudaDeviceSynchronize();`

(4 marks)

Answer

- `__syncthreads();` is used to synchronize threads within a block
  - Used to prevent RAW / WAR / WAW hazards

- `cudaDeviceSynchronize();` is used when we need the host to wait for all the threads to complete the kernels execution

4. (c) Describe how GPU is designed to facilitate the parallel programming paradigm in no more than 4 sentences (Hint: SIMD/SIMT).

(5 marks)

Answer

- GPU consists of many SM cores, each with many 'ALUs'
- Enable massive parallel MAC (floating points) operations
- Warp executes one common instruction for all its threads at a time
- Instructions are pipelined to achieve instruction-level parallelism
- Instructions are issued in order, no branch prediction and speculative execution
- Individual threads are free to branch and execute independently when the thread diverges

4. (d) Figure Q4d shows a CUDA kernel that runs on a GPU to compute the dot product of two vectors A and B, and output a scalar value C. Identify three GPU programming-related mistakes in the CUDA C code in Figure Q4b. Explain how they could be fixed.

$$C = \mathbf{A} \bullet \mathbf{B} = \sum_{i=1}^{N} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_N b_N$$

```
Line
1  void dot_prod(int N, int *a, int *b, int *c) {
2    int temp[N];
3    int i = threadIdx.x;
4    temp [i] = a[i]*b[i];
5
6    // Thread 0 sums the pairwise products
7    if (i == 0) {
8       int sum = 0;
9       for (int j = 0; j < N; j++)
10        sum += temp [j];
11      *c = sum;
12   }
13 }
```

Figure Q4b (6 marks)

Answer

Line 1: `__global__` is not used to declare the kernel function as device code

Line 2: `__shared__` is not used to declare the array in shared memory

Line 5: `__syncthreads()` is not used to synchronizes all threads within a block

Fixed code:

Line 1: `__global__ void dot_prod(int N, int *a, int *b, int *c){`

Line 3: `__shared__ int i = temp[N];`

Line 5: `__syncthreads();`

Appendix – Instruction Formats