MSc

Homework 2

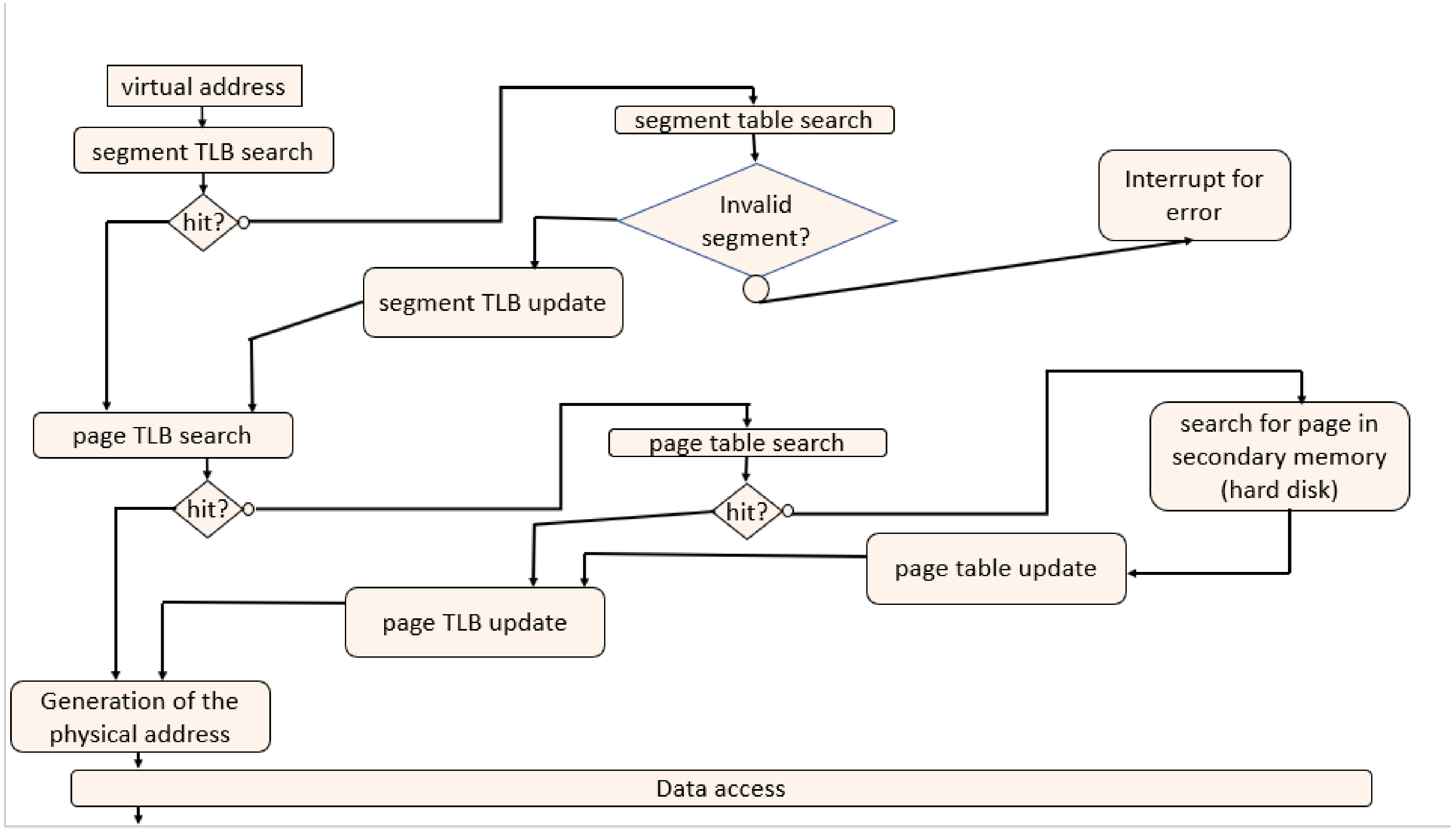
EXAM NUMBER: Y3874734

OLIVER FORBES-SHAW

Exam Number: Y3874734

Q1)

The diagram below, taken from the slides of lecture 10, illustrates the algorithm that controls the operation of pagination. Draw a similar algorithm for paged segmentation, assuming the presence of separate translation lookaside buffers (TLBs) for pages and segments. [12 mark



Exam Number: Y3874734

Q2)

- For a direct-mapped cache, compute:

- The number of bits required for the tag and for the index [1 mark].

Direct Mapped		
31 14	13 8	7 0
Tag 18 bits, 31 down to 14	Index 6 bits, 13 down to 8	Offset 8 bits, 7 down to 0

$Offset$ = 8 bits as shown below

$$2\ bits\ for\ byte\ addesable\ \left(2^2 = 4\ (number\ of\ bytes\ addressed)\right)$$

$$+$$

$$6\ bits\ for\ number\ of\ \ number\ of\ 32\ bit\ words\ in\ a\ block(2^6 = 64\ words)$$

$Index$ = 6 bits as shown below

$$6\ bits\ for\ number\ of\ slots\ for\ blocks\ \left(2^6 = 64\ (number\ of\ slots)\right)$$

Tag = 18 bits as shown below

Addresses are coded in 32 bits

$$Tag\ 18\ bits = 32 - 6(index) - 8(Offset)$$

- The number of cache hits and misses as the program completely reads the three arrays [2 marks].

Total hits = 252

All hits contained within array C.

Total misses = 516

A array has 4 total compulsory misses

B array has 4 total compulsory misses

C array has 4 total compulsory misses

A & B both have 504 conflict misses between them.

A and B arrays both have the same Index number (000110) so will always be a miss due to the way the system processes through the array. Due to the A and B containing the same index number the only one set is available so they conflict continuously, A overwriting B and vice versa this causes constant conflict misses for both arrays except for the compulsory misses. Array C has a different Index number so hits every time except for the 4 compulsory misses at the start of each new block of addresses.

- For a fully-associative cache, assuming a least-recently-used replacement policy, compute:

- The number of bits required for the tag (or address) [1 mark].

Fully associative cache	
31 8	7 0
Tag 24 bits, 31 down to 8	Offset 8 bits, 7 down to 0

$Offset$ $=$ 8 bits as shown below

$$2\ bits\ for\ byte\ addesable\ \left(2^2 = 4\ (number\ of\ bytes\ addressed)\right)$$

$$+$$

$$6\ bits\ for\ number\ of\ \ number\ of\ 32\ bit\ words\ in\ a\ block(2^6 = 64\ words)$$

$Index$ $=$ 0 bits as no Index in this method

Tag $=$ 24 bits as shown below

Addresses are coded in 32 bits

$$Tag\ 24\ bits = 32 - 8(Offset)$$

– The number of cache hits and misses as the program completely reads the three arrays [2 marks].

Total misses = 12

A array has 4 total compulsory misses

B array has 4 total compulsory misses

C array has 4 total compulsory misses

Total hits = 756

A array has 252 total hits

B array has 252 total hits

C array has 252 total hits

Due to each of the 12 blocks having different tag numbers they can all fit into the cache in different slots (4 for each array, 12 total of 64 available), there will be a total of 12 compulsory misses(4 for each array) but the hit to miss ratio is by far the best.

- For a set-associative cache with 2 blocks per set, assuming a not-last-used replacement policy, compute:

- The number of bits required for the tag and for the index [1 mark].

Set associative cache with 2 blocks per set		
31 13	12 8	7 0
Tag 19 bits, 31 down to 13	Index 5 bits, 12 down to 8	Offset 8 bits, 7 down to 0

$Offset$ = 8 bits as shown below

$$2\ bits\ for\ byte\ addesable\ \left(2^2 = 4\ (number\ of\ bytes\ addressed)\right)$$

$$+$$

$$6\ bits\ for\ number\ of\ \ number\ of\ 32\ bit\ words\ in\ a\ block(2^6 = 64\ words)$$

$Index$ = 5 bits as shown below

$$5\ bits\ for\ number\ of\ slots\ for\ blocks\ \left(2^5 = 32\ (number\ of\ slots)\right)$$

Tag = 19 bits as shown below

Addresses are coded in 32 bits

$$Tag\ 19\ bits = 32 - 5(index) - 8(Offset)$$

- The number of cache hits and misses as the program completely reads the three arrays [2 marks].

Total misses = 768

A array has 4 total compulsory misses

B array has 4 total compulsory misses

C array has 4 total compulsory misses

A array has 252 total conflict misses

B array has 252 total conflict misses

C array has 252 total conflict misses

The large amount of conflict misses is due to the fact that array A,B and C all have the same Index number and the not last used replacement policy in conjunction with the 2 blocks per slot as shown in the table on the next page. Due to there being only 2 blocks per slot and the same index number, one block will be over written each turn but because the last used replacement policy is implemented the next block that will be read and be a hit is over written every single time causing no hits what so ever.

array checked	set blocks currently held	last used	status hit/miss
A	XX	0,0	miss
B	AX	1,0	miss
C	AB	0,1	miss
A	CB	1,0	miss
B	CA	0,1	miss
C	BA	1,0	miss
A	BC	0,1	miss
B	AC	1,0	miss
C	AB	0,1	miss
A	CB	1,0	miss
B	CA	0,1	miss
C	BA	1,0	miss
A	BC	0,1	miss
B	AC	1,0	miss
C	AB	0,1	miss
A	CB	1,0	miss
B	CA	0,1	miss
C	BA	1,0	miss

- For a set-associative cache with 8 blocks per set, assuming a least-recently-used replacement policy, compute:

- The number of bits required for the tag and for the index [1 mark].

Set associative cache with **8** blocks per set		
31 11	10 8	7 0
Tag 21 bits, 31 down to 11	Index 3 bits, 10 down to 8	Offset 8 bits, 7 down to 0

$Offset$ = 8 bits as shown below

$$2\ bits\ for\ byte\ addesable\ \left(2^2 = 4\ (number\ of\ bytes\ addressed)\right)$$

$$+$$

$$6\ bits\ for\ number\ of\ \ number\ of\ 32\ bit\ words\ in\ a\ block(2^6 = 64\ words)$$

$Index$ = 3 bits as shown below

$$3\ bits\ for\ number\ of\ slots\ for\ blocks\ \left(2^3 = 8\ (number\ of\ slots)\right)$$

Tag = 21 bits as shown below

Addresses are coded in 32 bits

$$Tag\ 21\ bits = 32 - 3(index) - 8(Offset)$$

- The number of cache hits and misses as the program completely reads the three arrays [2 marks].

Total misses = 12

A array has 4 total compulsory misses

B array has 4 total compulsory misses

C array has 4 total compulsory misses

Total hits = 756

A array has 252 total hits

B array has 252 total hits

C array has 252 total hits

There are a large number of hits for this systems implementation due to the fact that each slot contains 8 blocks. There is a compulsory miss for each of the 12 blocks of 64 words of the A,B and C arrays being inserted into the slot and because each of the arrays index value(110) in the address is the same then they must be inserted into the same slot. Each slot has 8 blocks able to be held and 8 slots each so the blocks can hit each time excluding the compulsory ones. This means that the least-recently-used replacement policy does not cause any problems in this situation.

Q3)

Identify all the data (RAW only) and control hazards in this code for the 5-stage pipeline of Architecture D (see lecture 8), described in the lectures and copied below for reference [4 marks].

Data only hazard

03 loadr r3, r30 because of line 02 ,ori r30, r0 4

05 loado r4, r30, 3 because of line 02 ori r30, r0, 4

07 andi r5, r29, 16 because of line 04 , xori r29, r0, 65535

08 L3: andi r6, r4, 1 because of line 05 loado r4, r30

Data & control hazard

09 br r6, =0, +2 (L4) control hazard and data because of line 08 andi r6, r4, 1

14 br r5, ≠0, -6 (L3) control hazard and data because of 13 dec r5, r5 data hazard

To begin with, ignore the presence of any forwarding path (i.e., assume that there are no such paths) and assume that control hazards will be handled by the processor using pipeline stalls (i.e. no need to explicitly insert NOP instructions). Re-order the instructions to eliminate as many of the data hazards you have identified (without changing the function realized by the code, of course!). Then, insert the minimal number of NOP (no operation) instructions necessary to eliminate all remaining data dependencies. For your answer, rewrite the entire instructions and not just the line numbers. [7 marks].

Yellow indicates original data hazards

Original		
01		loadi r2, 16
02		ori r30, r0, 4
03		loadr r3, r30
04		xori r29, r0, 65535
05		loado r4, r30, 3
06		move r7, r0
07		andi r5, r29, 16
08	L3:	andi r6, r4, 1
09		br r6, =0, +2 (L4)
10		add r7, r3, r7
11	L4:	shr r4, r4, 1
12		shl r3, r3, 1
13		dec r5, r5
14		br r5, ≠0, -6 (L3)

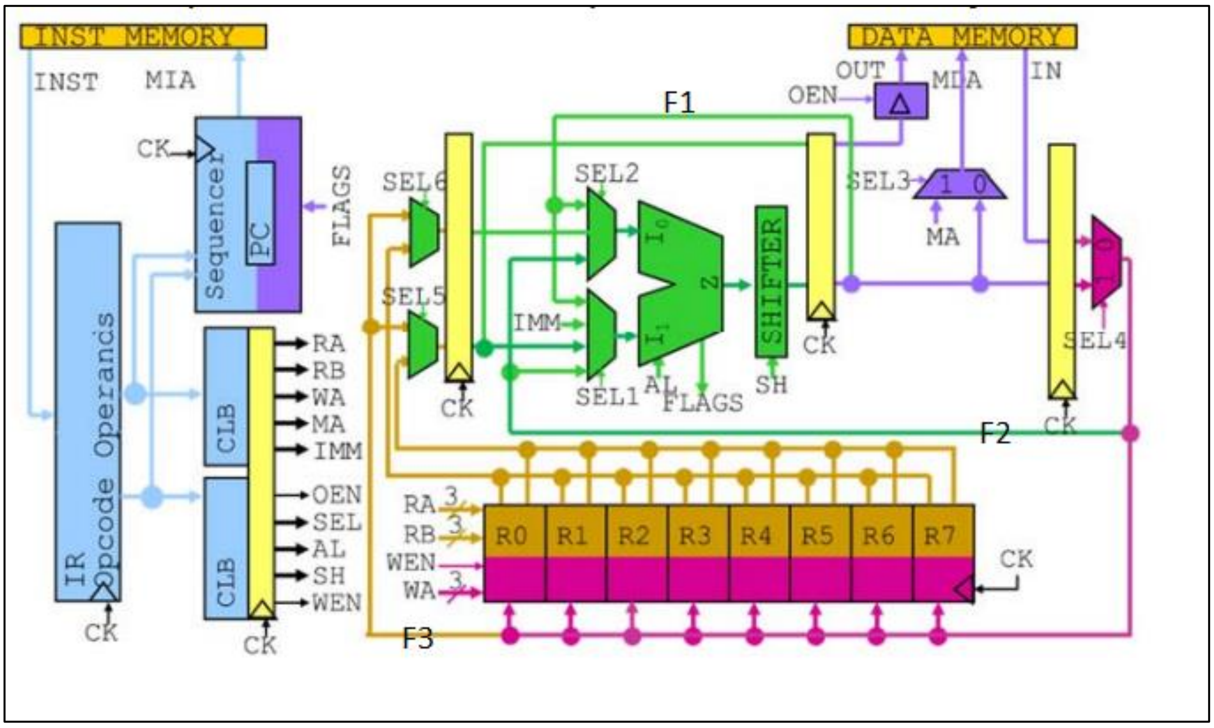
15		stori r7, 0

Green indicates NO Operation (NOP)

Old Instruction numbers	new instruction numbers	branch jump point	Instructions
02	01		ori r30, r0, 4
04	02		xori r29, r0, 65535
06	03		move r7, r0
02	04		loadi r2, 16
05	05		loado r4, r30, 3
	06		NOP
03	07		loadr r3, r30
07	08		andi r5, r29, 16
08	09	L3:	andi r6, r4, 1
	10		NOP
	11		NOP
	12		NOP
09	13		br r6, =0, +2 (L4)
10	14		add r7, r3, r7
13	15	L4:	dec r5, r5
12	16		shl r3, r3, 1
11	17		shr r4, r4, 1
15	18		stori r7, 0
14	19		br r5, ≠0, -6 (L3)

Exam Number: Y3874734

Go back to the original code (without reordering and NOPs). Let F1 be the forwarding path from the Memory Access stage to the Execute stage, F2 the path from the Register Write stage to the Execute stage, and F3 the path between the Register Write stage and the Register Read stage (see figure below as a reminder). Identify which (if any) of the bypass paths can be used to eliminate each data hazard you have identified [3 marks].



Data only hazard

03 loadr r3, r30 because of line 02 ,ori r30, r0 4 eliminated via the use of bypass path F1

05 loado r4, r30, 3 because of line 02 ori r30, r0, 4 eliminated via the use of bypass path F3

07 andi r5, r29, 16 because of line 04 , xori r29, r0, 65535 eliminated via the use of bypass path F3

08 L3: andi r6, r4, 1 because of line 05 loado r4, r30 eliminated via the use of bypass path F3

Data & control hazard

09 br r6, =0, +2 (L4) control hazard and data because of line 08 andi r6, r4, 1 eliminated via the use of bypass path F1

14 br r5, ≠0, -6 (L3) control hazard and data because of 13 dec r5, r5 data hazard eliminated via the use of bypass path F1

The code above has two conditional branch instructions: B1 at position 09 and B2 at position 14. For a multiplication of 16 bit data, each of those branches is evaluated 16 times. B1 is taken or not depending on the value of the single bits of the multiplier, and hence it can be assumed that it will be taken, on average, 50% of the time. B2 is used to select which bit of the multiplier is evaluated, and hence will be always taken the first 15 times, not taken the last (16th) time. For Architecture D and ignoring data hazards (i.e., assuming that forwarding will solve any issues related to data hazards), compute how many clock cycles would be necessary to execute the above piece of code:

- Without any form of speculative execution [3 marks];

Original			number of clk cycles	totals
01		loadi r2, 16	1	1
02		ori r30, r0, 4	1	1
03		loadr r3, r30	1	1
04		xori r29, r0, 65535	1	1
05		loado r4, r30, 3	1	1
06		move r7, r0	1	1
07		andi r5, r29, 16	1	1
08	L3:	andi r6, r4, 1	16	16
09		br r6, =0, +2 (L4)	16 x3	48
10		add r7, r3, r7	8	8
11	L4:	shr r4, r4, 1	16	16
12		shl r3, r3, 1	16	16
13		dec r5, r5	16	16
14		br r5, ≠0, -6 (L3)	16 x 3	48
15		stori r7, 0	1	1
		4clk cycles to finish	4	4
			total clk cycles =	180

Exam Number: Y3874734

- With speculative execution using static prediction – predict not taken [3 marks];

Original			number of clk cycles	totals
01		loadi r2, 16	1	1
02		ori r30, r0, 4	1	1
03		loadr r3, r30	1	1
04		xori r29, r0, 65535	1	1
05		loado r4, r30, 3	1	1
06		move r7, r0	1	1
07		andi r5, r29, 16	1	1
08	L3:	andi r6, r4, 1	16	16
09		br r6, =0, +2 (L4)	8x3, 8x1	32
10		add r7, r3, r7	8	8
11	L4:	shr r4, r4, 1	16	16
12		shl r3, r3, 1	16	16
13		dec r5, r5	16	16
14		br r5, ≠0, -6 (L3)	15x3, 1x1	46
15		stori r7, 0	1	1
		4clk cycles to finish	4	4
			total clk cycles =	162

- With speculative execution using static prediction – predict taken [3 marks];

Original			number of clk cycles	totals
01		loadi r2, 16	1	1
02		ori r30, r0, 4	1	1
03		loadr r3, r30	1	1
04		xori r29, r0, 65535	1	1
05		loado r4, r30, 3	1	1
06		move r7, r0	1	1
07		andi r5, r29, 16	1	1
08	L3:	andi r6, r4, 1	16	16
09		br r6, =0, +2 (L4)	8x3, 8x1	32
10		add r7, r3, r7	8	8
11	L4:	shr r4, r4, 1	16	16
12		shl r3, r3, 1	16	16
13		dec r5, r5	16	16
14		br r5, ≠0, -6 (L3)	15x1,1x3	18
15		stori r7, 0	1	1
		4clk cycles to finish	4	4
			total clk cycles =	134

- With speculative execution using static prediction – direction-based prediction [3 marks].

Original			number of clk cycles	totals
01		loadi r2, 16	1	1
02		ori r30, r0, 4	1	1
03		loadr r3, r30	1	1
04		xori r29, r0, 65535	1	1
05		loado r4, r30, 3	1	1
06		move r7, r0	1	1
07		andi r5, r29, 16	1	1
08	L3:	andi r6, r4, 1	16	16
09		br r6, =0, +2 (L4)	8x1, 3x8	32
10		add r7, r3, r7	8	8
11	L4:	shr r4, r4, 1	16	16
12		shl r3, r3, 1	16	16
13		dec r5, r5	16	16
14		br r5, ≠0, -6 (L3)	15x1, 1x3	18
15		stori r7, 0	1	1
		4clk cycles to finish	4	4
			total clk cycles =	134