Practice Midterm 2 Solutions

Prof. Y. Oruç's ENEE350, Fall 2010

University of Maryland

Prepared by: Tim Creech

10/30/2010

Problem 1: (30 points.)

(a)

See the microcode listing below:

```
ABUS=0;    BBUS=1; OBUS=ABUS+~BBUS+1; A=OBUS;     //A=-1
ABUS=*;    BBUS=1; OBUS=BBUS;         B=OBUS;     //B=1
ABUS=A;    BBUS=*; OBUS=ABUS+8;       A=OBUS;     //A=7; (call this A'.)
ABUS=*;    BBUS=B; OBUS=BBUS+8;       B=OBUS;     //B=9; (call this B'.)
ABUS=A;    BBUS=B; OBUS=ABUS+~BBUS+1; A=OBUS;     //A=A' - B'
ABUS=A;    BBUS=B; OBUS=ABUS+~BBUS+1; A=OBUS;     //A=A' - 2B'
ABUS=A;    BBUS=B; OBUS=ABUS+~BBUS+1; A=OBUS;     //A=A' - 3B'
ABUS=1;    BBUS=1; OBUS=ABUS+BBUS;    MAR=OBUS;   //MAR=2
ABUS=MAR;  BBUS=*; OBUS=ABUS+8;       MAR=OBUS;   //MAR=10
ABUS=MAR;  BBUS=*; OBUS=ABUS+8;       MAR=OBUS;   //MAR=18
ABUS=A;    BBUS=*; OBUS=ABUS;         MDR=OBUS;   //MDR=A
ABUS=1;    BBUS=*; OBUS=ABUS;         write=OBUS; //Memory[18]=MDR, or Memory[18]=A
```

The corresponding binary encodings follow immediately from the above listing and the reference table provided:

```
00 000 011 0100 0000
00 000 011 0001 0110
00 001 000 1001 0000
00 000 010 1010 0110
00 001 010 0100 0000
00 001 010 0100 0000
00 001 010 0100 0000
00 100 011 0010 0101
00 011 000 1001 0101
00 011 000 1001 0101
00 001 000 0000 0100
00 100 000 0000 1101
```

(b)

Note: Microstore addresses are in decimal, not hexadecimal.

Location	Microcode	Instr. Type	Explanation
0d100	01 0001 0001100111	Branch	If A==0, go to 103.
0d101	00 001 011 0100 0000	Data Transfer	`ABUS=A; BBUS=1; OBUS=ABUS-BBUS; A=OBUS;` (Subtract 1 from A.)
0d102	01 0011 0001100100	Branch	Unconditionally, go to 100.
0d103	00 000 011 0001 0111	Data Transfer	`RET_SET=1;` return.

This microprogram repeatedly subtracts 1 from A until A==0.

Problem 2: (20 points.)

(a)

Using the LRU eviction policy:

Block encountered:	1	7	6	6	5	4	3	3	3	5	2	1	1	7	5	6
Cache contents:	1	17	176	176	576	546	543	543	543	543	523	521	521	721	751	756
Cache hit?				✓				✓	✓	✓			✓			

(b)

Using the OPT eviction policy:

Block encountered:	1	7	6	6	5	4	3	3	3	5	2	1	1	7	5	6
Cache contents:	1	17	176	176	175	145	135	135	135	135	125	125	125	175	175	165
Cache hit?				✓				✓	✓	✓		✓	✓		✓	

(c)

The sequence is 16 accesses long. When using the LRU eviction strategy there were 5 hits, for a rate of $\frac{5}{16} \approx 0.3125$. When using OPT eviction there were 7 hits, for a rate of $\frac{7}{16} \approx 0.4375$. In this case, the OPT eviction policy performs better.

Problem 3: (30 points.)

Since each block/frame contains 64 bytes, let the lowest 6 bits of the accessed memory byte refer to the byte offset within the mapped frame. Since we have 8 sets, let the next 3 bits select the set from which the mapped frame is chosen. Finally, let the remaining 7 bits be the tag which will distinguish between the multiple blocks which could occupy a frame.

(a)

Since each set has 32 frames, this is a 32-way set-associative cache, and each address will potentially be mapped to any of the 32 frames.

(i) 1001 1011 1100 1111 - Here, the bits “111” indicate that this address’s block will be mapped to set 7’s frames.

(ii) 1101 1001 1101 1101 - Here, the bits “111” indicate that this address’s block will be mapped to set 7’s frames.

(iii) 1000 1111 1100 0111- Here, the bits “111” indicate that this address’s block will be mapped to set 7’s frames.

(b)

In this case, the 7-bit tag and 3-bit set address combine to create a 10-bit block address. This means that blocks “1001101111”, “1101100111”, and “1000111111” are being accessed. In decimal, these are blocks 623, 871, and 575. They all map to the same set, but the sets are large enough to hold all three blocks (and more). Assuming no evictions during the three accesses, the blocks in cache will be:

- 623 after (i)
- 623 and 871 after (ii)
- 623, 871, and 575 after (iii).

(c)

The following assumes that byte address accesses are performed uniformly at random over the entire range of the address space.

Since there are 8 sets and 1024 blocks, each set services 1024/8=128 blocks. Each of these 128 blocks can be assigned to any one of the 32 frames within the set. Note that block x would never be cached twice in the cache.

Due to the fact that 32 different blocks from a possible set of 128 blocks will be cached in any given block's mapped set, there is a probability of $\frac{32}{128} = 0.25$ that an accessed address will be within one of these 32 blocks and result in a hit.

The probability that there is a cache miss is equal to the probability that there is *not* a cache hit: $1-0.25$, or 0.75.

Problem 4: (20 points.)

The Vesp program will have the following logic:

1. Copy Mem[0x10] to B.
2. Compare B with 0x6c. If equal, jump to the 'l' routine.
3. Compare B with 0x73. If equal, jump to the 's' routine.
4. Compare B with 0x65. If equal, jump to the 'e' routine.
5. Compare B with 0x72. If equal, jump to the 'r' routine.
6. Compare B with 0x77. If equal, jump to the 'w' routine.
7. The input was not recognized, so jump to the command input routine.

See the Vesp code listing below:

```
3001 // Copy the command into B.
0010

2000 // Load the character 'l' to A.
006c
0300 // Get the difference between 'l' and A.
5uuu // Jump to the 'l' routine if the difference was 0.

2000 // Load the character 's' to A.
0073
0300 // Get the difference between 's' and A.
5vvv // Jump to the 's' routine if the difference was 0.

2000 // Load the character 'e' to A.
0065
0300 // Get the difference between 'e' and A.
5www // Jump to the 'e' routine if the difference was 0.

2000 // Load the character 'r' to A.
0072
0300 // Get the difference between 'r' and A.
5xxx // Jump to the 'r' routine if the difference was 0.

2000 // Load the character 'w' to A.
0077
0300 // Get the difference between 'w' and A.
5yyy // Jump to the 'w' routine if the difference was 0.

4zzz // Jump to ask the user to give valid input again, since we didn't
     //  recognize this input.
```