16.317: Microprocessor-Based Systems I

Spring 2012

Exam 2

April 4, 2012

Name: ________________________________ ID #: _________________ Section: _________

For this exam, you may use a calculator and one 8.5" x 11" double-sided page of notes. All other electronic devices (e.g., cellular phones, laptops, PDAs) are prohibited. If you have a cellular phone, please turn it off prior to the start of the exam to avoid distracting other students.

The exam contains 3 questions for a total of 100 points. Please answer the questions in the spaces provided. If you need additional space, use the back of the page on which the question is written and clearly indicate that you have done so.

The last four pages of the exam (beginning with page 7) contain reference material for the exam: lists of 80386 instructions and condition codes. You may detach these pages and do not have to submit them when you turn in your exam.

You will have 50 minutes to complete this exam.

Q1: Multiple choice	/ 20
Q2: Protected mode memory accesses	/ 40
Q3: Assembly language	/ 40
TOTAL SCORE	/ 100

1. (20 points, 5 points per part) ***Multiple choice***

For each of the multiple choice questions below, clearly indicate your response by circling or underlining the single choice you think best answers the question.

a. Given CS = 1000H, IP = E000, and EBX = 1E001000, which of the following CALL instructions will transfer control to an instruction at physical address 1F000H?

   A. CALL 1000H
   B. CALL F000H
   C. CALL BX
   D. CALL EBX
   E. CALL HOME_BECAUSE_YOUR_MOTHER_MISSES_YOU
      (hey, for all you know, that could be a valid instruction label)

   i. A and C

   ii. B and C

   iii. A and D

   iv. B and D

b. How many iterations does the following loop execute?

```
          MOV  CX, 0008H
          MOV  AX, 0000H
START:    ADD  AX, 0002H
          CMP  AX, CX
          LOOPNE START
```

   i. 2

   ii. 3

   iii. 4

   iv. 6

   v. 8

1 (cont.)

c. Assuming A, B, C, and D are all signed integers, what compound condition does the following instruction sequence test?

```
MOV     AX, D
CMP     A, AX
SETL    BL
SUB     AX, B
CMP     AX, C
SETGE   BH
AND     BL, BH
```

i. `(A < D) && (B >= C)`

ii. `(A < D) && (D >= C)`

iii. `(A <= D) && (D – B >= C)`

iv. `(A < D) && (D – B >= C)`

v. `(A <= D) && (B – D >= C)`

d. Which of the following statements about virtual memory are true?

A. When translating a virtual address to a physical address, the virtual page number is replaced by the appropriate physical frame number, while the lower bits of the address—the page offset—remain the same.

B. The number of bits in the page offset depends on the number of pages in the virtual address space.

C. Because all virtual pages cannot fit in physical memory, each page table entry requires a valid bit to indicate if the frame number in that entry is valid.

D. The TLB is a sandwich containing the same ingredients as a BLT, but with those ingredients stacked in the opposite order.

i. Only A

ii. Only C

iii. A and B

iv. A and C

v. A, B, and C

2. (40 points) ***Protected mode memory accesses***
Assume the 80386 is running in protected mode with the state given below. Note that each memory location shown contains a descriptor for a particular segment.

GDTR = 123000080017
LDTR = 0008
LDTR cache: base = 12300028
LDTR cache: limit = 0027

DS = 0006
ESI = 0000CD04
EBX = 00031A0

Memory	Address	**Memory**	Address

Base = 030010F0
Limit = 020F
	12300000	Base = AC000000
		Limit = 0317
			12300028

Base = 12300020
Limit = 0007
	12300008	Base = 01610200
		Limit = 03F7
			12300030

Base = 12300028
Limit = 0027
	12300010	Base = 03170214
		Limit = 030F
			12300038

Base = 1200C000
Limit = FFFF
	12300018	Base = 06B01000
		Limit = 0F07
			12300040

Base = 12340000
Limit = 00FF
	12300020	Base = 05000120
		Limit = 000F
			12300048

What address does each of the following instructions access? (Hint: solving part (a) should help you solve parts (b) and (c)).

a. `MOV     AX, [00H]`

b. `ADD     [SI], CX`

c. `SHL     [BX+10H], 7`

3. (40 points) ***Assembly language***

For each instruction sequence shown below, list all changed registers, memory locations, and/or flags, as well as their new values.

a. Initial state:

- (EAX) = 0000ABC0H
- (EBX) = 000012ACH
- (ECX) = 00000020H
- (EDX) = 00000000H
- (ESI) = 00000100H
- (EDI) = 00000200H
- (DS:100H) = 00H
- (DS:101H) = F0H
- (DS:110H) = 00H
- (DS:111H) = FFH
- (DS:200H) = 30H
- (DS:201H) = 00H
- (DS:210H) = AAH
- (DS:211H) = AAH
- (DS:220H) = 55H
- (DS:221H) = 55H
- (DS:300H) = AAH
- (DS:301H) = 55H

Also, assume all flags (ZF, CF, SF, PF, OF) are initialized to 0.

Instructions:

```
        BSF      DX, AX
        JNZ      END
        BT       BX, DX
        SETNC    [100H]
END:    AND      CL, [100H]
```

3 (cont.)

b. Initial state:

- (EAX) = 00000016H
- (EBX) = 00000317H
- (ECX) = 00000010H
- (EDX) = 0000ABCDH
- (ESI) = 00000100H
- (EDI) = 00000106H
- (DS:100H) = 0FH
- (DS:101H) = F0H
- (DS:102H) = 00H
- (DS:103H) = FFH
- (DS:104H) = 30H
- (DS:105H) = 00H
- (DS:106H) = AAH
- (DS:107H) = AAH
- (DS:108H) = 55H
- (DS:109H) = 55H
- (DS:10AH) = AAH
- (DS:10BH) = 55H

Also, assume all flags (ZF, CF, SF, PF, OF) are initialized to 0.

Instructions:

```
      CMP      AX, BX
      JE       L1
      JG       L2
      INC      AX
      JMP      END
L1:   DEC      AX
      JMP      END
L2:   MOV      AX, BX
END:  MOV      [DI+02H], AX
```

The following pages contain references for use during the exam: tables containing the 80386 instruction set and condition codes. You may detach these sheets from the exam and do not need to submit them when you finish.

Remember that:

- Most instructions can have at most one memory operand.
- Brackets [ ] around a register name, immediate, or combination of the two indicates an effective address. That address is in the data segment unless otherwise specified.
    - Example: MOV AX, [10H] → contents of DS:10H moved to AX
- Parentheses around a logical address mean "the contents of memory at this address".
    - Example: (DS:10H) → the contents of memory at logical address DS:10H

Category	**Instruction**	**Example**	**Meaning**
Data transfer	Move	`MOV AX, BX`	`AX = BX`
	Move & sign-extend	`MOVSX EAX, DL`	`EAX = DL, sign-extended to 32 bits`
	Move and zero-extend	`MOVZX EAX, DL`	`EAX = DL, zero-extended to 32 bits`
	Exchange	`XCHG AX, BX`	`Swap contents of AX, BX`
	Load effective address	`LEA AX, [BX+SI+10H]`	`AX = BX + SI + 10H`

	Load full pointer	`LDS AX, [10H]`
		
		`LSS EBX, [100H]`
			`AX = (DS:10H)`
			`DS = (DS:12H)`
			
			`EBX = (DS:100H)`
			`SS = (DS:104H)`

Arithmetic	Add	`ADD AX, BX`	`AX = AX + BX`
	Add with carry	`ADC AX, BX`	`AX = AX + BX + CF`
	Increment	`INC [DI]`	`(DS:DI) = (DS:DI) + 1`
	Subtract	`SUB AX, [10H]`	`AX = AX – (DS:10H)`
	Subtract with borrow	`SBB AX, [10H]`	`AX = AX – (DS:10H) – CF`
	Decrement	`DEC CX`	`CX = CX – 1`
	Negate (2's complement)	`NEG CX`	`CX = -CX`

	Unsigned multiply (all operands are non-negative, regardless of MSB value)	`MUL BH`
		`MUL CX`
		`MUL DWORD PTR [10H]`
			`AX = BH * AL`
			`(DX,AX) = CX * AX`
			`(EDX,EAX) = (DS:10H) * EAX`

	Signed multiply (all operands are signed integers in 2's complement form)	`IMUL BH`
		`IMUL CX`
		`IMUL DWORD PTR[10H]`
			`AX = BH * AL`
			`(DX,AX) = CX * AX`
			`(EDX,EAX) = (DS:10H) * EAX`

	Unsigned divide	`DIV BH`
		
		`DIV CX`
		
		`DIV EBX`
			`AL = AX / BH (quotient)`
			`AH = AX % BH (remainder)`
			
			`AX = EAX / CX (quotient)`
			`DX = EAX % CX (remainder)`
			
			`EAX = (EDX,EAX) / EBX (Q)`
			`EDX = (EDX,EAX) % EBX (R)`

Category	Instruction	Example	Meaning
Logical	Logical AND	`AND AX, BX`	`AX = AX & BX`
	Logical inclusive OR	`OR AX, BX`	`AX = AX | BX`
	Logical exclusive OR	`XOR AX, BX`	`AX = AX ^ BX`
	Logical NOT (1's complement)	`NOT AX`	`AX = ~AX`

Shift/rotate (NOTE: for all instructions except RCL/RCR, CF = last bit shifted out)	Shift left	`SHL AX, 7`
		
		`SAL AX, CX`
			`AX = AX << 7`
			
			`AX = AX << CX`

	Logical shift right (treat value as unsigned, shift in 0s)	`SHR AX, 7`	`AX = AX >> 7`
			`(upper 7 bits = 0)`

	Arithmetic shift right (treat value as signed; maintain sign)	`SAR AX, 7`	`AX = AX >> 7`
			`(upper 7 bits = MSB of original value)`

	Rotate left	`ROL AX, 7`	`AX = AX rotated left by 7 (lower 7 bits of AX = upper 7 bits of original value)`
	Rotate right	`ROR AX, 7`	`AX=AX rotated right by 7 (upper 7 bits of AX = lower 7 bits of original value)`
	Rotate left through carry	`RCL AX, 7`	`(CF,AX) rotated left by 7 (Treat CF & AX as 17-bit value with CF as MSB)`

	Rotate right through carry	`RCR AX, 7`	`(AX,CX) rotated right by 7`
			`(Treat CF & AX as 17-b8t value with CF as LSB)`

Bit test/ scan	Bit test	`BT AX, 7`	`CF = Value of bit 7 of AX`

	Bit test and reset	`BTR AX, 7`	`CF = Value of bit 7 of AX`
			`Bit 7 of AX = 0`

	Bit test and set	`BTS AX, 7`	`CF = Value of bit 7 of AX`
			`Bit 7 of AX = 1`

	Bit test and complement	`BTC AX, 7`	`CF = Value of bit 7 of AX`
			`Bit 7 of AX is flipped`

	Bit scan forward	`BSF DX, AX`	`DX = index of first non-zero bit of AX, starting with bit 0`
			`ZF = 0 if AX = 0, 1 otherwise`

	Bit scan reverse	`BSR DX, AX`	`DX = index of first non-zero bit of AX, starting with MSB`
			`ZF = 0 if AX = 0, 1 otherwise`

Category	Instruction	Example	Meaning
Flag control	Clear carry flag	`CLC`	`CF = 0`
	Set carry flag	`STC`	`CF = 1`
	Complement carry flag	`CMC`	`CF = ~CF`
	Clear interrupt flag	`CLI`	`IF = 0`
	Set interrupt flag	`STI`	`IF = 1`
	Load AH with contents of flags register	`LAHF`	`AH = FLAGS`

	Store contents of AH in flags register	`SAHF`	`FLAGS = AH`
			`(Updates SF,ZF,AF,PF,CF)`

Conditional tests	Compare	`CMP AX, BX`	`Subtract AX – BX`
			`Updates flags`

	Byte set on condition	`SETcc AH`	`AH = FF if condition true`
			`AH = 0 if condition false`

Jumps and loops	Unconditional jump	`JMP label`	`Jump to label`
	Conditional jump	`Jcc label`	`Jump to label if condition true`
	Loop	`LOOP label`	`Decrement CX; jump to label if CX != 0`

	Loop if equal/zero	`LOOPE label`
		`LOOPZ label`
			`Decrement CX; jump to label if (CX != 0) && (ZF == 1)`

	Loop if not equal/zero	`LOOPNE label`
		`LOOPNZ label`
			`Decrement CX; jump to label if (CX != 0) && (ZF == 0)`

Subroutine-related instructions	Call subroutine	`CALL label`	`Jump to label; save address of instruction after CALL`
	Return from subroutine	`RET label`	`Return from subroutine (jump to saved address from CALL)`

	Push	`PUSH AX`
		
		`PUSH EAX`
			`SP = SP – 2`
			`(SS:SP) = AX`
			
			`SP = SP – 4`
			`(SS:SP) = EAX`

	Pop	`POP AX`
		
		`POP EAX`
			`AX = (SS:SP)`
			`SP = SP + 2`
			
			`EAX = (SS:SP)`
			`SP = SP + 4`

	Push flags	`PUSHF`	`Store flags on stack`
	Pop flags	`POPF`	`Remove flags from stack`
	Push all registers	`PUSHA`	`Store all general purpose registers on stack`
	Pop all registers	`POPA`	`Remove general purpose registers from stack`

Condition code	Meaning	Flags
O	Overflow	OF = 1
NO	No overflow	OF = 0
B NAE C	Below Not above or equal Carry	CF = 1
NB AE NC	Not below Above or equal No carry	CF = 0
S	Sign set	SF = 1
NS	Sign not set	SF = 0
P PE	Parity Parity even	PF = 1
NP PO	No parity Parity odd	PF = 0
E Z	Equal Zero	ZF = 1
NE NZ	Not equal Not zero	ZF = 0
BE NA	Below or equal Not above	CF OR ZF = 1
NBE A	Not below or equal Above	CF OR ZF = 0
L NGE	Less than Not greater than or equal	SF XOR OF = 1
NL GE	Not less than Greater than or equal	SF XOR OF = 0
LE NG	Less than or equal Not greater than	(SF XOR OF) OR ZF = 1
NLE G	Not less than or equal Greater than	(SF XOR OF) OR ZF = 0