16.317: Microprocessor-Based Systems I

Summer 2012

Exam 2
August 1, 2012

Name: _____ **ID #:** _____

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 two hours 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. Assume SS = 3000H and SP = F018H before the 80386 executes the following instructions:

```
PUSH    AX
PUSH    CX
PUSH    EDX
PUSH    ESI
```

What is the physical address of the top of the stack after executing the instructions above?

i. 30000H

ii. 3F00CH

iii. 3F010H

iv. 3F018H

v. 3F024H

b. You are given the incomplete loop below:

```
        MOV  CX, 000AH
        MOV  SI, FFFFH
L:      INC  SI
        MOV  AX, [SI]
        CMP  AX, 00H
        _______________
```

Choose one of the instructions below to fill in the blank so that the loop above will exit if (a) 10 iterations have been completed, or (b) the `MOV` instruction loads a non-zero byte from memory:

i. JMP L

ii. LOOP L

iii. LOOPE L

iv. LOOPNE L

v. IMUL AX

1 (cont.)

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

```
MOV     AX, A
ADD     AX, B
CMP     AX, C
SETLE   BL
MOV     AX, C
CMP     AX, D
SETG    BH
OR      BL, BH
```

i. `(A <= C) || (C > D)`

ii. `(B <= C) || (C > D)`

iii. `(A + B <= C) || (C >= D)`

iv. `(A <= B + C) || (C > D)`

v. `(A + B <= C) || (C > D)`

d. Which of the following statements accurately reflect your opinion(s)? Circle all that apply (but please don't waste too much time on this question)!

i. "I still don't know what the difference between a selector and a descriptor is."

ii. "I'm not sure Dr. Geiger knows what the difference between a selector and a descriptor is."

iii. "Would someone please explain why we're not just programming in C?"

iv. "Is the semester over yet?"

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 = 001631A00038
LDTR = 0010
LDTR cache: base = 00163180
LDTR cache: limit = 001F

DS = 000E
ES = 001B
EDI = 0000444A
EBX = 0000F000

Memory	Address
Base = 030010F0 Limit = 020F	00163170
Base = 12300020 Limit = 0007	00163178
Base = A0331010 Limit = 0027	00163180
Base = FE002200 Limit = FFFF	00163188
Base = 12340000 Limit = 00FF	00163190

Memory	Address
Base = AC000000 Limit = 0317	00163198
Base = 01610200 Limit = 03F7	001631A0
Base = 00163170 Limit = 0027	001631A8
Base = 00163180 Limit = 001F	001631B0
Base = 05000120 Limit = C00F	001631B8

What address does each of the following instructions access?

a. `MOV DX, [40H]`

b. `XOR ES:[DI], CX`

c. `BSF AX, WORD PTR [BX+100H]`

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: 00000012H
EDI: 00000200H
DS: 4130H
FLAGS: 00H

Address				
41300H	00	F0	08	00
41304H	10	10	00	FF
41308H	30	00	19	91
4130CH	20	40	60	80
41310H	AA	AA	AB	0F
41314H	00	16	55	55
41318H	17	03	7C	EE
4131CH	AA	55	42	D2
41320H	86	75	30	90

Instructions:

```
BTC       BX, 6


SETNC  DL


BSR       AX, [SI]


AND       AH, DL


SAHF
```

3 (cont.)
b. Initial state:

EAX: 00003170H
EBX: 0000315CH
ECX: 000031C5H
EDX: 00000000H
ESI: 00000012H
EDI: 0000001CH
DS: 4130H
FLAGS: 00H

Address				
41300H	00	F0	08	00
41304H	10	10	00	FF
41308H	30	00	19	91
4130CH	20	40	60	80
41310H	AA	AA	AB	0F
41314H	00	16	55	55
41318H	17	03	7C	EE
4131CH	AA	55	42	D2
41320H	86	75	30	90

Notes: For CMP instructions, note the relationship between compared values (e.g., "AX < BX"). For jumps, indicate if the jump is taken and why (e.g., "JG not taken because AX < BX"). Only evaluate instructions that are actually executed—don't evaluate skipped instructions.

Instructions:

```
        CMP     AX, BX

        JL      L1

        CMP     AX, CX

        JL      L2

        INC     AX

        JMP     END

L1:     DEC     AX

        JMP     END

L2:     MOV     AX, 0123H

END:    MOV     [DI], 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