The following pages contain references for use during the exam: a table containing the 80386 instructions we have covered thus far. 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,CF) rotated right by 7`
			`(Treat CF & AX as 17-b8t value with CF as LSB)`