The following pages contain references for use during the exam: tables containing the x86 instruction set (covered so far) and condition codes. You do not need to submit these pages when you finish your exam.

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.
  - Example: MOV AX, [0x10] → contents of address 0x10 moved to AX
- Parentheses around an address mean "the contents of memory at this address".
  - Example: (0x10) → the contents of memory at address 0x10

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+0x10]`	`AX = BX + SI + 0x10`
Arithmetic	Add	`ADD AX, BX`	`AX = AX + BX`
	Add with carry	`ADC AX, BX`	`AX = AX + BX + CF`
	Increment	`INC [EDI]`	`(EDI) = (EDI) + 1`
	Subtract	`SUB AX, [0x10]`	`AX = AX – (0x10)`
	Subtract with borrow	`SBB AX, [0x10]`	`AX = AX – (0x10) – CF`
	Decrement	`DEC CX`	`CX = CX – 1`
	Negate (2's complement)	`NEG CX`	`CX = -CX`

	Multiply
	Unsigned: `MUL`
	(all operands are non-negative)
	Signed: `IMUL`
	(all operands are signed integers in 2's complement form)
		`IMUL BH`
		
		`IMUL CX`
		
		`MUL DWORD PTR [0x10]`
			`AX = BH * AL`
			
			`(DX,AX) = CX * AX`
			
			`(EDX,EAX) = (0x10) * EAX`

	Divide
	Unsigned: `DIV`
	(all operands are non-negative)
	Signed: `IDIV`
	(all operands are signed integers in 2's complement form)
		`DIV BH`
		
		`IDIV 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 (bit flip)	`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 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
Conditional tests	Compare	`CMP AX, BX`	`Subtract AX - BX` `Updates flags`
	Byte set on condition	`SETcc AH`	`AH = 1 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`
			`(SP) = AX`
			
			`SP = SP - 4`
			`(SP) = EAX`

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

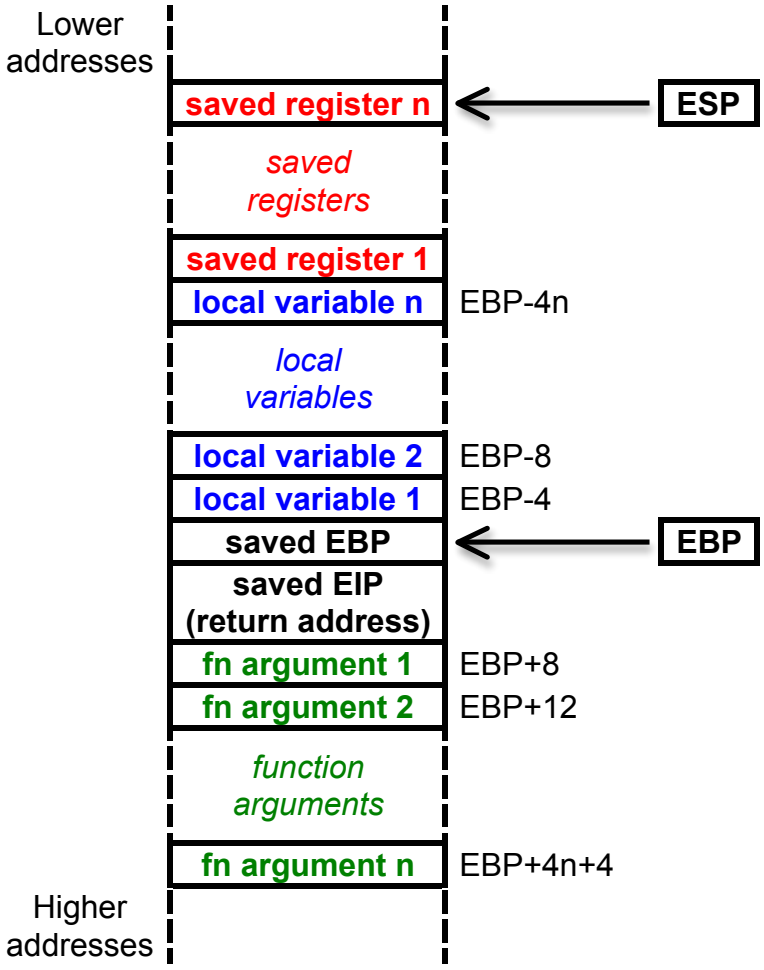
	Push flags	`PUSHF`	`Store flags on stack`
	Pop flags	`POPF`	`Remove flags from stack`
	Push all registers	`PUSHA`	`Store all general purpose registers on stack`
	Pop all registers	`POPA`	`Remove general purpose registers from stack`

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

x86 subroutine details:

- Subroutine arguments are passed on the stack, and can be accessed within the body of the subroutine starting at address EBP+8.
- At the start of each subroutine:
  - Save EBP on the stack
  - Copy the current value of the stack pointer (ESP) to EBP
  - Create space within the stack for each local variable by subtracting the appropriate value from ESP. For example, if your function uses four integer local variables, each of which contains four bytes, subtract 16 from ESP. Local variables can then be accessed starting at the address EBP-4.
  - Save any registers the function uses other than EAX, ECX, and EDX.
- A subroutine's return value is typically stored in EAX.

Typical x86 stack frame (covered in HLL → assembly lectures)



The following two pages contain tables listing the PIC 16F1829 instruction set. You do not need to submit these pages with your exam.

Remember that, in the table below:

- f = a register file address
- W = the working register
- d = destination select: “F” for a file register, “W” for the working register
- b = bit position within an 8-bit file register
- k = literal field, constant data or label
- PC = the program counter
- C = the carry bit in the status register
- Z = the zero bit in the status register

TABLE 29-3: PIC16(L)F1825/1829 ENHANCED INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C, DC, Z	2
ADDWFC	f, d	Add with Carry W and f	1	11	1101	dfff	ffff	C, DC, Z	2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	2
ASRF	f, d	Arithmetic Right Shift	1	11	0111	dfff	ffff	C, Z	2
LSLF	f, d	Logical Left Shift	1	11	0101	dfff	ffff	C, Z	2
LSRF	f, d	Logical Right Shift	1	11	0110	dfff	ffff	C, Z	2
CLRF	f	Clear f	1	00	0001	1fff	ffff	Z	2
CLRW	–	Clear W	1	00	0001	0000	00xx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	2
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	2
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff		2
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C, DC, Z	2
SUBWFB	f, d	Subtract with Borrow W from f	1	11	1011	dfff	ffff	C, DC, Z	2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	2
BYTE ORIENTED SKIP OPERATIONS									
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1, 2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1, 2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		2
BIT-ORIENTED SKIP OPERATIONS									
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		1, 2
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		1, 2
LITERAL OPERATIONS									
ADDLW	k	Add literal and W	1	11	1110	kkkk	kkkk	C, DC, Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLB	k	Move literal to BSR	1	00	0000	001k	kkkk		
MOVLP	k	Move literal to PCLATH	1	11	0001	1kkk	kkkk		
MOVLW	k	Move literal to W	1	11	0000	kkkk	kkkk		
SUBLW	k	Subtract W from literal	1	11	1100	kkkk	kkkk	C, DC, Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

TABLE 29-3: PIC16(L)F1825/1829 ENHANCED INSTRUCTION SET (CONTINUED)

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
CONTROL OPERATIONS						
BRA	k	Relative Branch	2	11 001k kkkk kkkk		
BRW	–	Relative Branch with W	2	00 0000 0000 1011		
CALL	k	Call Subroutine	2	10 0kkk kkkk kkkk		
CALLW	–	Call Subroutine with W	2	00 0000 0000 1010		
GOTO	k	Go to address	2	10 1kkk kkkk kkkk		
RETFIE	k	Return from interrupt	2	00 0000 0000 1001		
RETLW	k	Return with literal in W	2	11 0100 kkkk kkkk		
RETURN	–	Return from Subroutine	2	00 0000 0000 1000		
INHERENT OPERATIONS						
CLRWDT	–	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$, $\overline{PD}$	
NOP	–	No Operation	1	00 0000 0000 0000		
OPTION	–	Load OPTION_REG register with W	1	00 0000 0110 0010		
RESET	–	Software device Reset	1	00 0000 0000 0001		
SLEEP	–	Go into Standby mode	1	00 0000 0110 0011	$\overline{TO}$, $\overline{PD}$	
TRIS	f	Load TRIS register with W	1	00 0000 0110 0fff		
C-COMPILER OPTIMIZED						
ADDFSR	n, k	Add Literal k to FSRn	1	11 0001 0nkk kkkk		
MOVIW	n mm	Move Indirect FSRn to W with pre/post inc/dec modifier, mm	1	00 0000 0001 0nmm	Z	**2, 3**
	k[n]	Move INDFn to W, Indexed Indirect.	1	11 1111 0nkk kkkk	Z	**2**
MOVWI	n mm	Move W to Indirect FSRn with pre/post inc/dec modifier, mm	1	00 0000 0001 1nmm		**2, 3**
	k[n]	Move W to INDFn, Indexed Indirect.	1	11 1111 1nkk kkkk		**2**

Source for table: *"PIC16(L)F1825/1829 Data Sheet", Microchip Technology, Inc. http://ww1.microchip.com/downloads/en/DeviceDoc/41440C.pdf*