CPE 390: Microprocessor Systems

Lecture 1
Introduction

Matthew Conforth

Dept. of Electrical and Computer Engineering
Stevens Institute of Technology

Course Introduction

- Instructor: Matthew Conforth
  - Email: Matthew.Conforth@stevens.edu
  - Office hours: After class or by appointment
- Course Website:
  - http://390.revan.us
- Laboratory website:
  - Course website -> Lab Syllabus
  - First lab starts from the second week.

Course Introduction

- Textbooks
  - Primary
    Daniel J. Pack & Steven F. Barrett, *Microcontroller Theory and Applications: HC12 & S12, Second Edition*, Pearson Prentice Hall, ISBN 978-0-13-615205-7, 2008
  - Supplemental (Optional)
    H-W. Huang, *MC68HC12: An Introduction,* Thomson Delmar Learning, ISDN 0-7668-3448-4, 2003
- Student Assessment
  - Lecture: 75%
    - within the lecture portion, 30% hw, 30% midterm, 40% final exam
  - Lab: 25%, including lab wrap-up report and final project
  - Class participation: “swing factor”

Course Introduction

- Homework
  - One homework will be due per class, starting week 2
  - See the course website for details
  - The solutions for the homework assignments will be posted online after the due date
  - Your grade will be reduced for late submission
  - Homework will not be accepted after the solution is posted
  - You must do your homework independently
  - Copying homework from any source is cheating

Basic Computer Concepts

- What is a computer?
  - Hardware and software
- Computer hardware organization

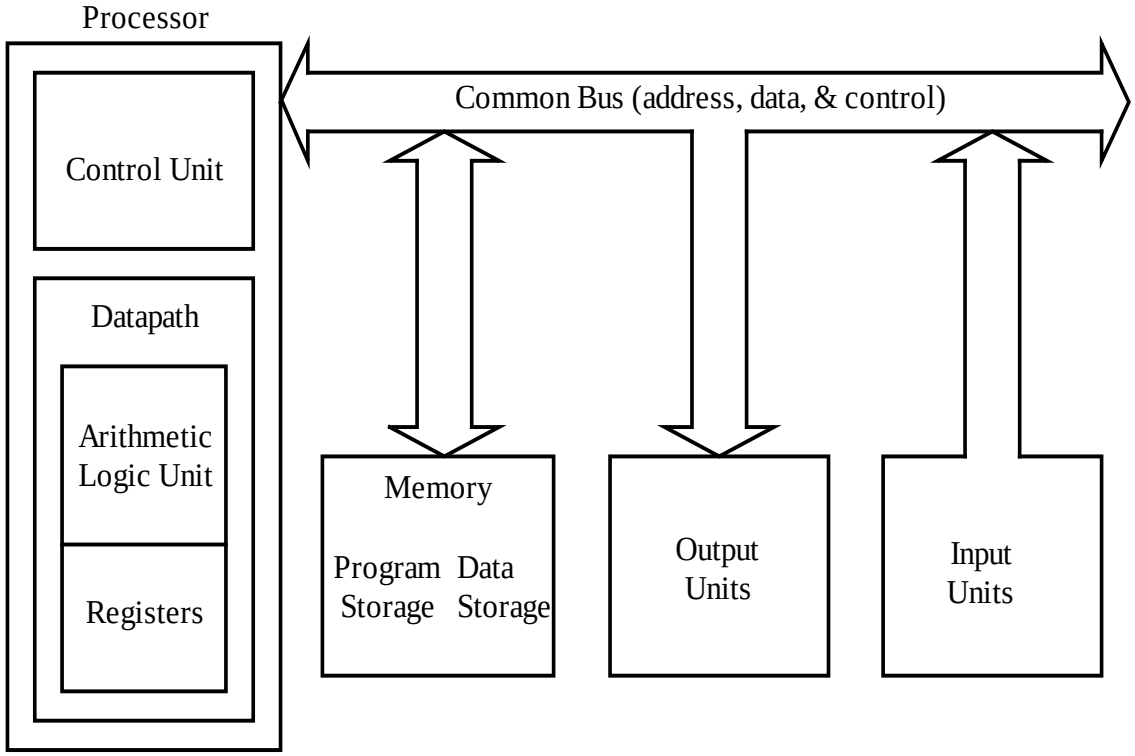


Figure 1.1 Computer Organization

Basic Computer Concepts

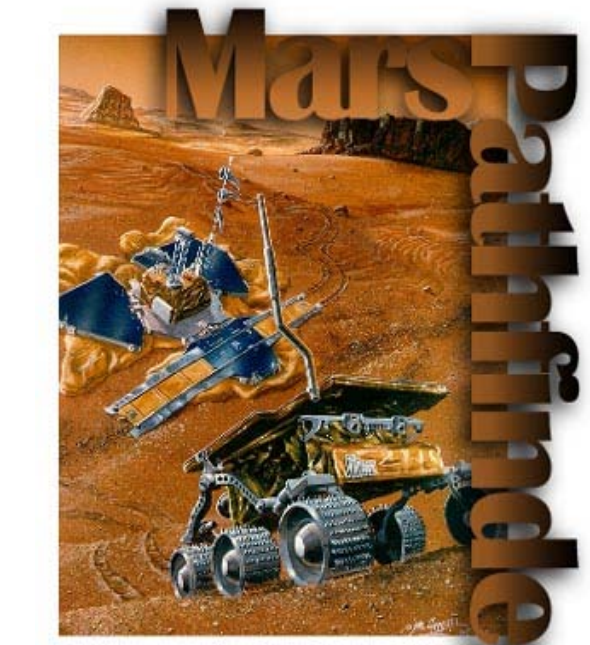
- The processor: also called central processing unit (CPU)
  - Datapath
    - Register file : storage location in the CPU
    - Arithmetic logic unit (ALU) : arithmetic computations and logic evaluations
  - Control unit: contains hardware instruction logic
    - Program counter (PC): controls the memory address of the next instruction to be executed
    - Status register: flags the instruction execution result
- The microprocessor
  - A processor fabricated on a single integrated circuit
  - Peripheral chips are needed to construct a product
- The microcontroller
  - The processor and peripheral functions implemented on one VLSI chip
  - Contains one or more components besides microprocessor: memory, timer, ADC, DAC, DMA, parallel port, serial port, memory component interface circuitry, and DSP feature.

Main Features of 68HC12

- 16-bit CPU
- 64 kB memory space
- 768 bytes to 4 kB of EEPROM
- 1 kB to 12 kB of on-chip SRAM
- 32 kB to 128 kB flash memory
- Sophisticated timer functions that include: input capture, output compare, pulse accumulators, and real-time interrupt
- Serial communication interfaces: SCI, SPI
- Background debug mode (BDM)
- 8-bit or 10-bit A/D converter
- Instructions for supporting fuzzy logic function

Embedded Systems

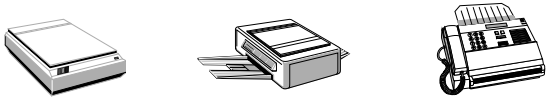
- Embedded computing systems
  - Computing systems embedded within electronic devices with tightly coupled hardware and software integration
  - Designed to perform a dedicated function
  - Usually an integral part of a larger system
  - Multiple embedded systems can coexist in an embedded system

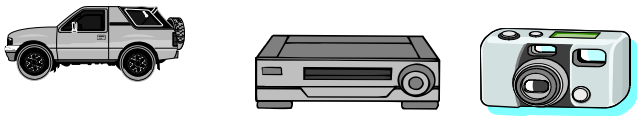


Embedded Systems Examples

Sony's Aibo

Sonicare toothbrush





almost everywhere in our life ….

Palm
Handheld

GPS

Common Characteristics of Embedded Systems

- Single-functioned
  - Executes a single program, repeatedly
- Tightly-constrained
  - Low cost, low power, small, fast, etc.
- Reactive and real-time
  - Continually reacts to changes in the system's environment
  - Must compute certain results in real-time without delay

Memory

- Semiconductor memory
  - **Random access memory** (RAM): same amount of time is required to access any location on the same chip
  - **Read-only memory** (ROM): can only be read; cannot be written to by the processor

- Random access memory : volatile
  - **Dynamic random access memory** (DRAM): periodic refresh is required to maintain the contents of a DRAM chip
  - **Static random access memory** (SRAM): no periodic refresh is required

- Read-only memory : nonvolatile
  - **Mask-programmed read-only memory** (MROM): programmed when being manufactured
  - **Programmable read-only memory** (PROM): the memory chip can be programmed by the end user

Memory

Erasable Programmable ROM (EPROM)

1. Electrically programmable many times
2. Erased by ultraviolet light (through a window)
3. Erasable in bulk (whole chip in one erasure operation)

Electrically Erasable Programmable ROM (EEPROM)

1. Electrically programmable many times
2. Electrically erasable many times
3. Can be erased one location, one row, or whole chip in one operation
4. The circuit design of EEPROM is required to erase the memory

Flash Memory

1. Electrically programmable many times
2. Electrically erasable many times
3. Can only be erased in block or the entire chip
4. No dedicated programmer is required

Computer's Software

Computer Software

- Computer programs are known as software
- A program is a sequence of instructions

Machine Instruction

- A sequence of binary digits that can be executed by the processor, for example, 0100 0011.
- Hard to understand, program, and debug for human being

Assembly Language

- Defined by assembly instructions
- An assembly instruction is a mnemonic representation of a machine instruction
- Assembly programs must be translated before it can be executed -- translated by an assembler
- Programmers need to work on the program logic at a very low level and can not achieve high productivity.

Computer's Software

High-level Language

- Syntax of a high-level language is similar to English
- A translator is required to translate the program written in a high-level language -- done by a compiler
- High-level languages allow the user to work on the program logic at higher level

Source Code

- A program written in assembly or high-level language

Object Code

- The output of an assembler or compiler

Cross Assembler / Cross Compiler

- Runs on one computer but generates machine instructions to be run on a different computer with a totally different instruction set

A list file example

line	addr.	machine code	source code	
1:		= 00001000	org	$1000
2:	1000	B6 0800	ldaa	$800
3:	1003	BB 0801	adda	$801
4:	1006	BB 0802	adda	$802
6:	1009	7A 0900	staa	$900
			end	

Registers

- CPU registers
  - General purpose operations, such as arithmetic, logic, and program flow control
- I/O registers
  - Configure the operations of peripheral functions, hold data transferred in and out of the I/O ports, and record the status of I/O operations
  - Classified into *data, data direction, control, and status registers*

68HC12 CPU Registers

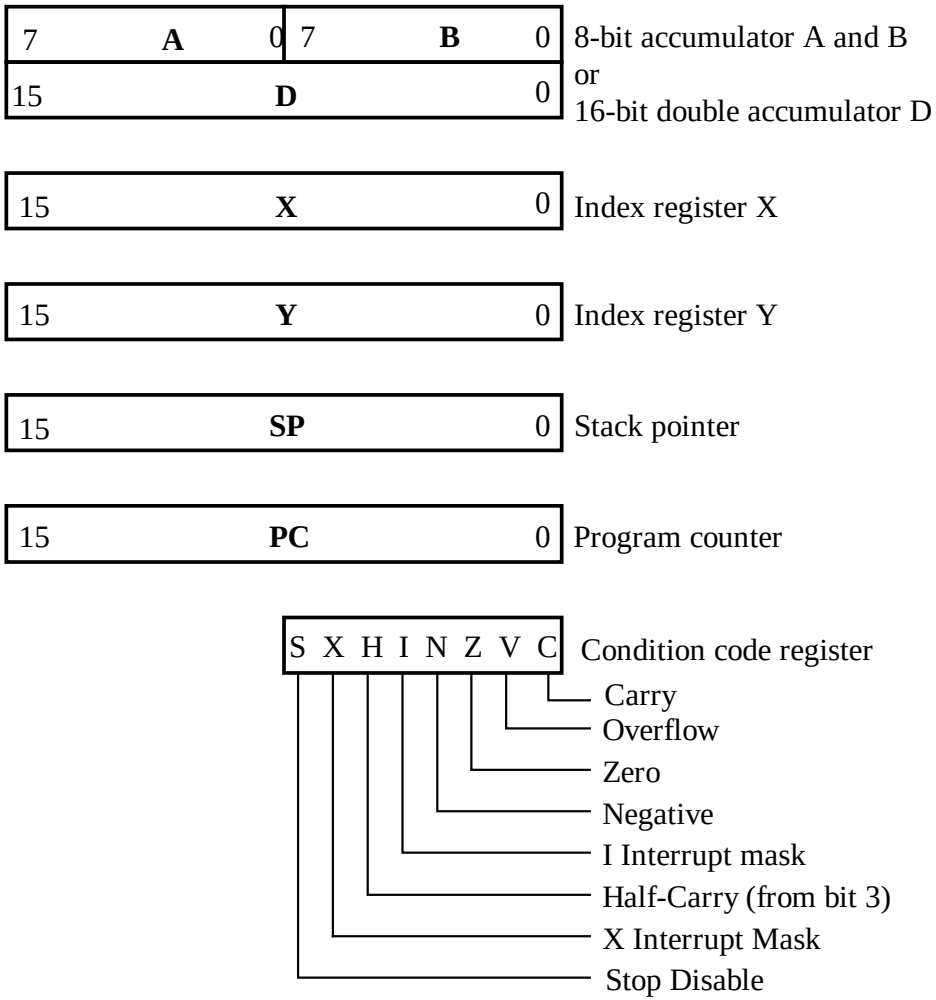
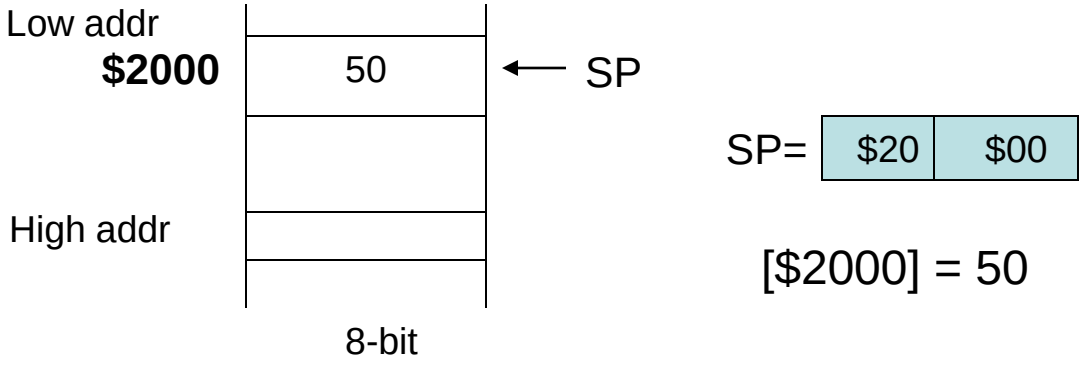


Figure 1.3 MC68HC12 CPU registers.

Registers

- Data registers: A, B, D
  - A = Dh, B= Dl
- Memory address registers: X, Y, SP
  - All 16 bits registers
  - X,Y allow you to address memory location with offset
  - SP: point to the top byte of the stack, the stack grows toward lower address



Registers

- PC register
  - 16-bit
  - Always point to the next instruction to be executed
  - After each execution of the instructions, PC is incremented by the number of bytes of the executed instruction
- Condition code register (CCR)
  - 8-bit
  - Used in condition branch and interrupt handling

Number Representation

- 68HC12 supports several unsigned number formats
  - bit — can represent [0, 1]
  - 8-bit unsigned integer — can represent range [0, 255]
  - 16-bit unsigned integer — can represent range [0, 65535]
  - 32-bit unsigned integer — can represent range [0, 4294967295]
- Multi-byte numbers are stored in Big Endian

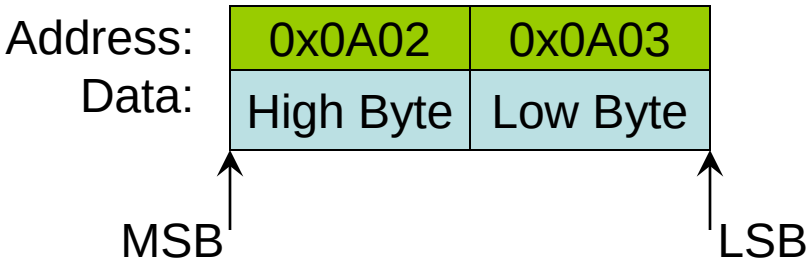
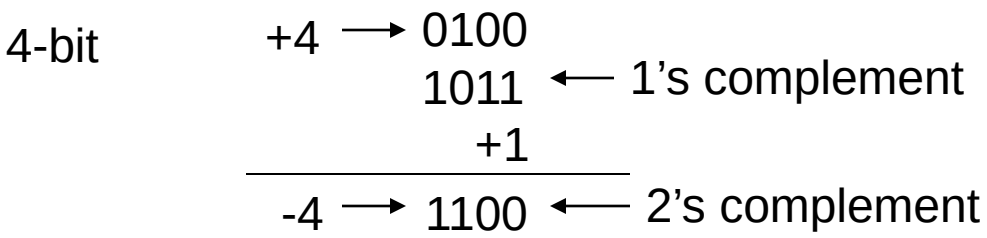


Table 1.1 Prefixes for number bases

base	prefix	example
binary	%	%10001010
octal	@	@1234567
decimal		12345678
hexadecimal (shorthand **hex**)	$	$5678

Number Representation

- 68HC12 uses two's complement format to represent signed numbers
  - MSB is a sign bit (0 – positive, 1 – negative)
  - 5-bit signed integer can represent range [-16, +15]
  - **8-bit signed integer** **can represent range [-128, +127]**
  - 9-bit signed integer can represent range [-256, +255]
  - **16-bit signed integer** **can represent range [-32768, +32767]**
  - 32-bit signed integer can represent range [-2147483648, +2147483648]
  - Example:



Memory Addressing

- Memory consists of a sequence of directly addressable **locations**.
- A location is referred to as an **information unit**.
- A memory location can be used to store **data**, **instruction**, and the **status** of peripheral devices.
- A memory location has two components: an **address** and its **contents**

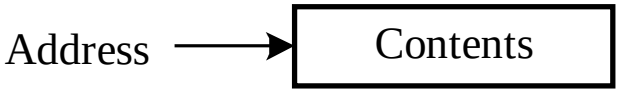
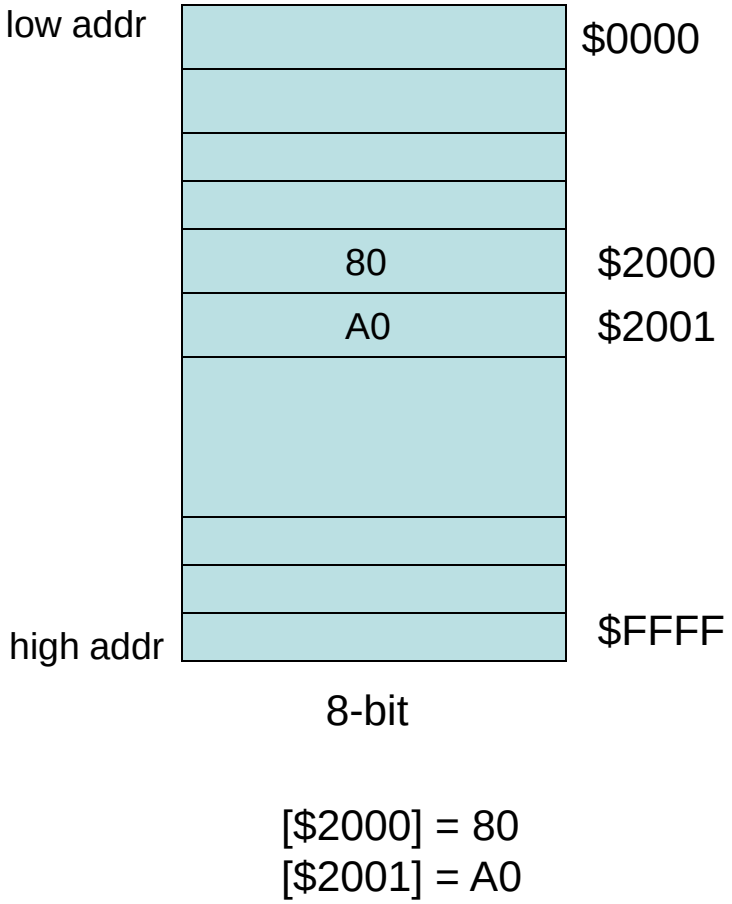


Figure 1.5 The components of a memory location

Memory Addressing



68HC12 supports 64KB address space
16-bit address bus and 16-bit data bus

← Effective address : the actual memory address

[$2000] = 80
[$2001] = A0

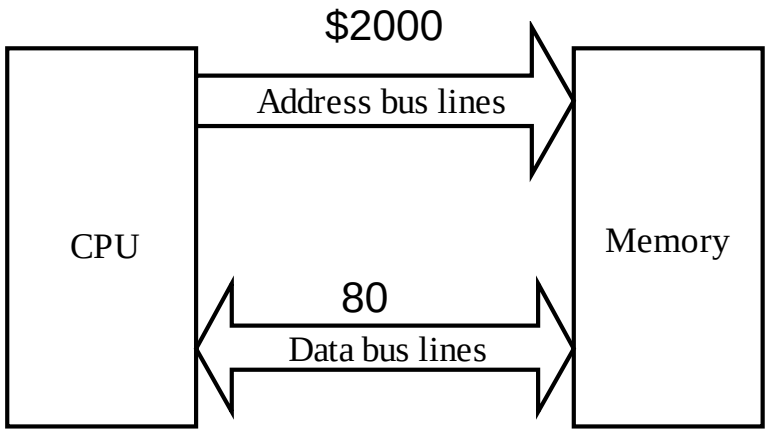


Figure 1.6 Transferring data between CPU and memory

68HC12 Addressing Mode

- A 68HC12 instruction consists of one or two bytes of **opcode** and zero to five bytes of **operand** addressing information.

- Opcode bytes specify the operation to be performed by the CPU. The first byte of a two-byte opcode is always $18.

- Addressing modes determines how the CPU access memory locations to be acted upon.

Table 1.2P M68HC12 addressing mode summary

Addressing mode	Source format	Abbre.	Description
Inherent	**INST** (no externally supplied operands)	INH	Operands (if any) are in CPU registers
Immediate	**INST** #opr8i or **INST** #opr16i	IMM	Operand is included in instruction stream. 8- or 16-bit size implied by context
Direct	**INST** opr8a	DIR	Operand is the lower 8 bits of an address in the range $0000-$00FF
Extended	**INST** opr16a	EXT	Operand is a 16-bit address
Relative	**INST** rel8 or **INST** rel16	REL	An 8-bit or 16-bit relative offset from the current PC is supplied in the instruction
Indexed (5-bit offset)	**INST** oprx5,xysp	IDX	5-bit signed constant offset from x,y,sp, or pc
Indexed (pre-decrement)	**INST** oprx3,-xys	IDX	Auto pre-decrement x, y, or sp by 1 ~ 8
Indexed (pre-increment)	**INST** oprx3,+xys	IDX	Auto pre-increment x, y, or sp by 1 ~ 8
Indexed (post-decrement)	**INST** oprx3,xys-	IDX	Auto post-decrement x, y, or sp by 1 ~ 8
Indexed (post-increment)	**INST** oprx3,xys+	IDX	Auto post-increment x, y, or sp by 1 ~ 8
Indexed (accumulator offset)	**INST** abd,xysp	IDX	Indexed with 8-bit (A or B) or 16-bit (D) accumulator offset from x, y, sp, or pc
Indexed (9-bit offset)	**INST** oprx9,xysp	IDX1	9-bit signed constant offset from x, y, sp, or pc (lower 8-bits of offset in one extension byte)
Indexed (16-bit offset)	**INST** oprx16,xysp	IDX2	16-bit constant offset from x, y, sp, or pc (16-bit offset in two extension bytes)
Indexed-Indirect (16-bit offset)	**INST** [oprx16,xysp]	[IDX2]	Pointer to operand is found at 16-bit constant offset from (x, y, sp, or pc)
Indexed-Indirect (D accumulator offset)	**INST** [**D**,xysp]	[D,IDX]	Pointer to operand is found at x, y, sp, or pc plus the vlaue in D

Addressing Mode

Inherent Mode (INH)

- Instructions that use this mode do not uses extra bytes to specify operands because the instructions either do not need operands or all operands are CPU registers.
- Operands are implied by the opcode.
- Examples:

```
NOP          ; no operation
INX          ; X <- X+1
DECA         ; A <- A -1
```

Immediate Mode (IMM)

- Operands for instructions that use immediate mode are included in the instruction.
- CPU does not access memory for operands.
- Examples:

```
LDAA    #$55      ; A <- $55
LDX     #$800     ; X <- $800
```

Addressing Mode

Direct Mode (DIR)

- This mode can only specify memory locations in the range of 0– 255 ($00 - $FF), page zero.
- This mode uses only one byte to specify the operand address.
- Examples:

```
LDAA   $20    ; A<- m[$20]
LDX    $20    ; Xh<- m[$20], Xl<-m[$21]
```

Extended Mode (EXT)

- In this mode, the full 16-bit address is provided in the instruction.
- Examples:

```
LDAA   $4000    ; A<-m[$4000]
LDX    $FE60    ; Xh<-m[$FE60], Xl<-m[$FE61]
```

Addressing Mode

Relative Mode (REL)

- Used only by branch instructions.
- 8-bit offset: -128 ~ +127.
- 16-bit offset. -32768 ~ +32767.
- A programmer uses a symbol to specify the branch target and the assembler will figure out the actual branch offset (distance) from the instruction that follows branch instruction.

- For example,

```
$ B201   BGE  MINUS   ; if >= 0, branch to MINUS
                      ;    PC <- PC + branch offset
$ B202   ADDA  $10    ; else continue
            …
MINUS   DECB          ;
```

Addressing Mode

Index Mode

- This mode uses the sum of an index register (X, Y, PC, or SP) and an offset to specify the address of an operand.
- The offset can be a 5-bit, 9-bit, and 16-bit signed value or the value in accumulator A, B, or D.
- Automatic pre- or post-increment or pre- or post-decrement by -8 to +8 are options.
- PC can be used as the index register for all but auto-increment or auto-decrement mode.
- Indirect indexing with 16-bit offset or accumulator D as the offset is supported.
- A summary of indexed addressing modes is given in Table 1.3.

Table 1.3 Summary of indexed operations

Postbyte code (xb)	source code syntax	**Comments** rr: 00 = X, 01 = Y, 10 = SP, 11 = PC
rr0nnnnn	r n,r -n,r	**5-bit constant** offset n = -16 to +15 r can be X, Y, SP, or PC
111rr0zs	n,r -n,r	**Constant offset** (9- or 16-bit signed) z: 0 = 9-bit with sign in LSB of postbyte (s) -256< n < 255 1 = 16-bit 0 < n < 65535 if z = s = 1, 16-bit offset indexed-indirect (see below) r can be X, Y, SP, or PC
111rr011	[n,r]	**16-bit offset indexed-indirect** 0 < n < 65536 rr can be X, Y, SP, or PC
rr1pnnnn	n,-r n,+r n,r- n,r+	**Auto pre-decrement/increment or auto post-decrement/increment**; p = pre-(0) or post-(1), n = -8 to -1 or +1 to +8 r can be X, Y, or SP (PC not a valid choice) +8 = 0111 ... +1 = 0000 -1 = 1111 .... -8 = 1000
111rr1aa	A,r B,r D,r	**Accumulator offset** (unsigned 8-bit 0r 16-bit) aa: 00 = A 01 = B 10 = D (16-bit) 11 = see accumulator D offset indexed-indirect r can be X, Y, SP, or PC
111rr111	[D,r]	**Accumulator D offset indexed-indirect** r can be X, Y, SP, or PC

Addressing Mode

5-bit Constant Offset Indexed Addressing

- The base index register can be X, Y, SP, or PC.
- The range of the offset: -16 ~ +15.
- Examples,

```
LDX  #$2000   ;
LDAA  1,X     ;  A<- [$2000 + 1]
LDAB  -5,X    ;  B <- [$2000 – 5]=[$1FFB]
```

9-bit Constant Offset Indexed Addressing

- The base index register can be X, Y, SP, or PC.
- The range of the offset is from -256 to +255.
- Examples,

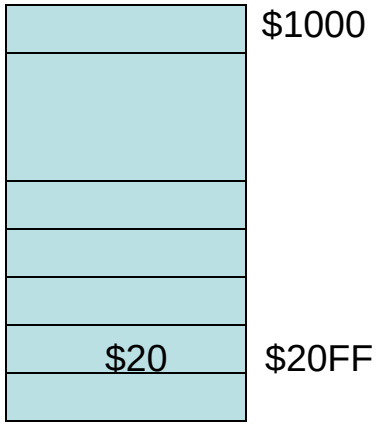
```
LDX #$2000
LDY #$3000
LDAA  $FF,X   ; A<- [$2000+$FF] =[$20FF]
LDAB  -20,Y   ;  B<- [$3000-20]
```

Addressing Mode

16-bit Constant Offset Indexed Addressing

- Range: 64KB ($0000 - $FFFF)
- Examples:

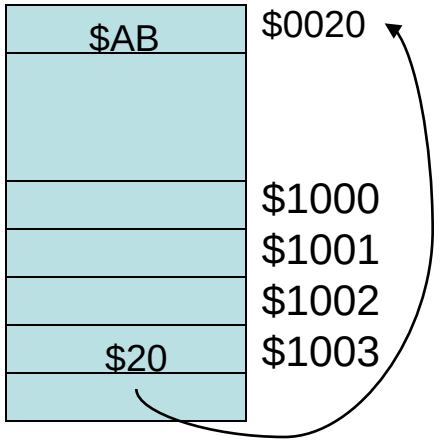
```
LDX  #$1000
LDAA  $10FF,X       ; A <- [$1000+$10FF]=[$20FF]
                    ;  A <- $20
```



16-bit Constant Indirect Indexed Addressing

```
LDY     #$1000
LDAA   [3, Y]        ; A <- $AB
```

Square brackets is used to distinguish this addressing mode from 16-bit constant offset indexed mode



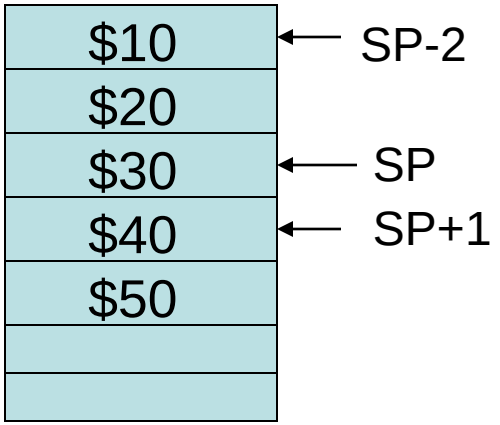
Addressing Mode

Auto Pre/Post Decrement/Increment Indexed Addressing

- The base index register can be X, Y, or SP.
- The index register can be incremented or decremented by an integer value either before or after indexing taking place.
- The index register retains the changed value after indexing.
- range: -8 through -1 or 1 through 8.

- Examples:

```
LDAB  2, -SP     ; B<- $10, SP<- SP-2
LDAA  1, SP+     ; A<- $30, SP<-SP+1
```



Addressing Mode

Accumulator Offset Indexed Addressing

- The effective address of the operand is the sum of the accumulator and the base index register.
- The base register can be X, Y, SP, or PC.
- The accumulator can be the 8-bit A or B or the 16-bit accumulator D.

- Examples:

```
LDAA  B,X    ;  A<- [ X + B ]
```

Addressing Mode

Accumulator D Indirect Indexed Addressing

- [D + index register] -> the memory location which contains a pointer to the memory location affected by the instruction
- For example,

```
            jmp     [D,PC]
go1         dc.w    target1     ; the DC.W reserves two bytes to hold
go2         dc.w    target2     ; the value of the symbol that follows
go3         dc.w    target3
            …
target1     …
            .
target2     …
            .
target3     …
```

68HC12 Instruction Examples

The LOAD and STORE Instructions

- The LOAD instruction copies the contents of a memory location or places an immediate value into an accumulator or a CPU register.
- STORE instructions save the contents of a CPU register into a memory location.
- N and Z flags of the CCR register are automatically updated.
- All except for the relative mode can be used to select the memory location or value to be loaded into an accumulator or CPU register.
- All except for the relative and immediate modes can be used to select memory location to store contents of the CPU register. For example,

```
LDAA  0,X
STAA  $20
STX   $8000
LDD    #100
```

Table 1.4 Load and store instructions

Mnemonic	Function	Operation
LDAA	Load A	(M) ⇒A
LDAB	Load B	(M) ⇒ B
LDD	Load D	(M:M+1) ⇒ (A:B)
LDS	Load SP	(M:M+1) ⇒ SP
LDX	Load index register X	(M:M+1) ⇒ X
LDY	Load index register Y	(M:M+1) ⇒ X
LEAS	Load effective address into SP	Effective address ⇒ SP
LEAX	Load effective address into X	Effective address ⇒ X
LEAY	Load efective address into Y	Effective address ⇒ Y
Store Instructions		
Mnemonic	Function	Operation
STAA	Store A	(A) ⇒ M
STAB	Store B	(B) ⇒ M
STD	Store D	(A) ⇒ M, (B) ⇒ M+1
STS	Store SP	(SP) ⇒ M, M+1
STX	Store X	(X) ⇒ M:M+1
STY	Store Y	(Y) ⇒ M:M+1

68HC12 Instruction Examples

Transfer and Exchange Instructions

- Transfer instructions copy the contents of a CPU register or accumulator into another CPU register or accumulator.
- **TFR** is the universal transfer instruction, but other mnemonics are accepted for compatibility with the 68HC11.
- The TAB and TBA instructions affect the N, Z, and V condition code bits.
- The **TFR** instruction does not affect any condition code bits. For example,

```
TFR  D,X          ; [D] ⇒ X
TFR  A,B          ; [A] ⇒ B
```

- The **EXG** instruction exchanges the contents of a pair of registers or accumulators. For example,

```
exg  A, B
exg  D,X
```

- The **SEX** instruction sign-extend an 8-bit two's complement number into a 16-bit number so that it can be used in 16-bit signed operations. For example,

```
SEX    A,X  ; copy the contents of A to the lower byte of X and
```

duplicate the bit 7 of A to every bit of the upper byte of X

68HC12 Instruction Examples

Move Instructions

- These instructions move data bytes or words from a source to a destination in memory.
- Six combinations of immediate, extended, and index addressing modes are allowed to specify the source and destination addresses:

  IMM ⇒ EXT, IMM ⇒ IDX, EXT ⇒ EXT,

  EXT ⇒ IDX, IDX ⇒ EXT, IDX ⇒ IDX

- Examples:

```
movb    $100,$800    ; [$100]->[$800]
movw    0,X, 0,Y     ; [X] -> [Y]
```

Table 1.6 Move instructions

Transfer Instructions		
Mnemonic	Function	Operation
MOVB MOVW	Move byte (8-bit) Move word (16-bit)	(M1) ⇒ M2 (M:M+1_1) ⇒ M:M+1_2

68HC12 Instruction Examples

Add and Subtract Instructions

-These instructions perform fundamental arithmetic operations.

-The destinations of these instructions are always a CPU register or accumulator.

-There are two-operand and three-operand versions of these instructions.

-Three-operand ADD or SUB instructions always include the C flag as one of the operand.

-Three-operand ADD or SUB instructions are used to perform multi-precision addition or subtraction.

-For example,

```
ADDA $800        ; A ⇐ [A] + [$800]
ADCA $800        ; A ⇐ [A] + [$800] + C
SUBA $802        ; A ⇐ [A] - [$802]
SBCA $800        ; A ⇐ [A] - [$800] - C
```

Table 1.7 Add and subtract instructions

Add Instructions		
Mnemonic	Function	Operation
ABA	Add B to A	(A) + (B) ⇒ A
ABX	Add B to X	(B) + (X) ⇒ X
ABY	Add B to Y	(B) + (Y) ⇒ Y
ADCA	Add with carry to A	(A) + (M) + C ⇒ A
ADCB	Add with carry to B	(B) + (M) + C ⇒ B
ADDA	Add without carry to A	(A) + (M) ⇒ A
ADDB	Add without carry to B	(B) + (M) ⇒ B
ADDD	Add without carry to D	(A:B) + (M:M+1) ⇒ A:B
Subtract Instructions		
Mnemonic	Function	Operation
SBA	Subtract B from A	(A) - (B) ⇒ A
SBCA	Subtract with borrow from A	(A) - (M) - C ⇒ A
SBCB	Subtract with borrow from B	(B) - (M) - C ⇒ B
SUBA	Subtract memory from A	(A) - (M) ⇒ A
SUBB	Subtract memory from B	(B) - (M) ⇒ B
SUBD	Subtract memory from D	(D) - (M:M+1) ⇒ D

Instruction Execution Cycle

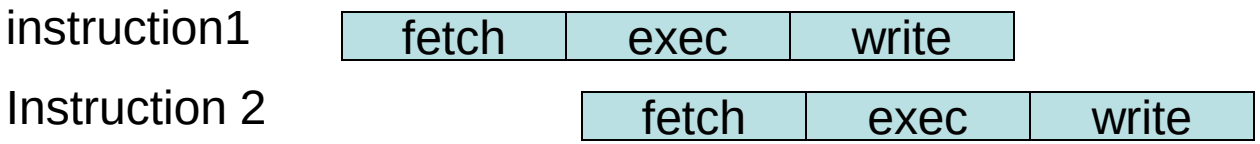
- Read cycle: access memory location, i.e., fetch instructions (opcodes) or operands
- Write cycle: store a value into memory location
- Instruction execution cycle: execute an instruction

- 68HC12 instructions requires
  - One or more read cycles to fetch opcode and address info
  - One or more read cycles to fetch operand (if have any)
  - One or more write cycle to either a register or memory location
- 68HC12 execute one instruction at a time, many may take several clock cycle to complete

- Program info is fetched in aligned 16-bit words

Instruction Queue

- The 68HC12 executes one instruction at a time and many instructions take several clock cycles to complete.
- When the CPU is performing the operation, it does not need to access memory.
- The 68HC12 prefetches instructions when the CPU is not accessing memory to speed up the instruction execution process (pipeline).



- There are two 16-bit queue stages and one 16-bit buffer. Unless buffering is required, program information is first queued in stage 1, and then advanced to stage 2 for execution.

Homework #1

- See course website: http://390.revan.us
  - click homework tab
- Please submit a hard copy