Microprocessor/Microcontroller

Introduction

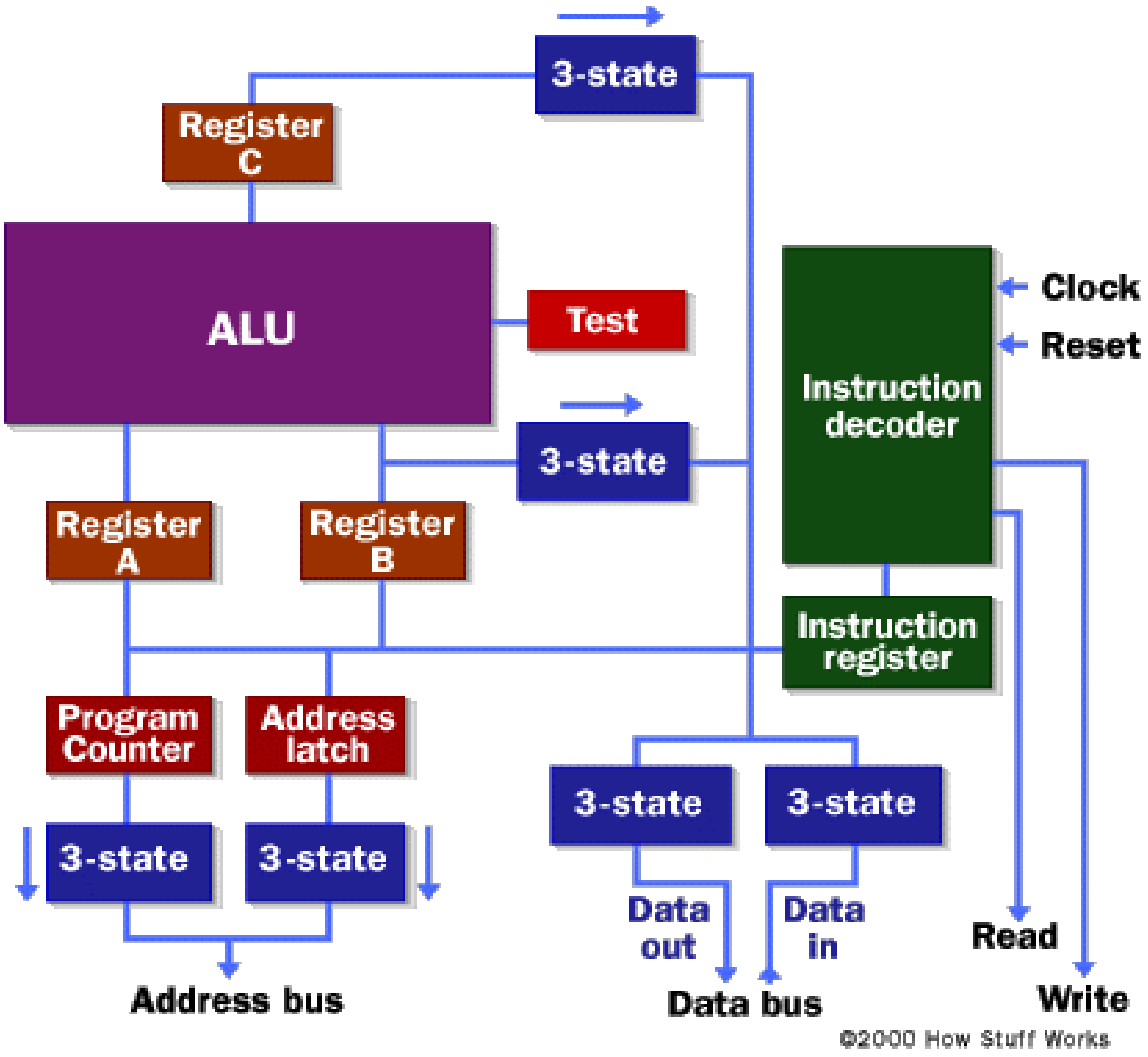
Microprocessor/Microcontroller

A microprocessor - also known as a **CPU** or central processing unit - is a complete computation engine that is fabricated on a single chip. The first microprocessor was the Intel 4004, introduced in 1971. The 4004 was not very powerful - all it could do was add and subtract, and it could only do that 4 bits at a time. But it was amazing that everything was on one chip. Prior to the 4004, engineers built computers either from collections of chips or from discrete components (Transistors and such). The 4004 powered one of the first portable electronic calculators. (excerpts from **How Microprocessors Work** by Marshall Brain)

Microprocessor/Microcontroller

- The first microprocessor to make it into a home computer was the Intel 8080, a complete 8-bit computer on one chip, introduced in 1974. The first microprocessor to make a real splash in the market was the Intel 8088, introduced in 1979 and incorporated into the IBM PC (which first appeared around 1982). If you are familiar with the PC market and its history, you know that the PC market moved from the 8088 to the 80286 to the 80386 to the 80486 to the Pentium to the Pentium II to the Pentium III to the Pentium 4. All of these microprocessors are made by Intel and all of them are improvements on the basic design of the 8088. The Pentium 4 can execute any piece of code that ran on the original 8088, but it does it about 5,000 times faster! (excerpts from **How Microprocessors Work** by Marshall Brain)

Microprocessor/Microcontroller



Microprocessor/Microcontroller

- This is about as simple as a microprocessor gets. This microprocessor has:
- An **address bus** (that may be 8, 16 or 32 bits wide) that sends an address to memory
- A **data bus** (that may be 8, 16 or 32 bits wide) that can send data to memory or receive data from memory
- An **RD** (read) and **WR** (write) line to tell the memory whether it wants to set or get the addressed location
- A **clock line** that lets a clock pulse sequence the processor
- A **reset line** that resets the program counter to zero (or whatever) and restarts execution
- Let's assume that both the address and data buses are 8 bits wide in this example.

Microprocessor/Microcontroller

Here are the components of this simple microprocessor:

Registers A, B and C are simply latches made out of flip-flops.

The address latch is just like registers A, B and C.

The program counter is a latch with the extra ability to increment by 1 when told to do so, and also to reset to zero when told to do so.

Microprocessor/Microcontroller

Microprocessor Instructions

- Even the incredibly simple microprocessor shown in the previous example will have a fairly large set of instructions that it can perform. The collection of instructions is implemented as bit patterns, each one of which has a different meaning when loaded into the instruction register. Humans are not particularly good at remembering bit patterns, so a set of short words are defined to represent the different bit patterns. This collection of words is called the **assembly language** of the processor. An **assembler** can translate the words into their bit patterns very easily, and then the output of the assembler is placed in memory for the microprocessor to execute.

Microprocessor/Microcontroller

Here's the set of assembly language instructions that the designer might create for the simple microprocessor in our example:

- **LOADA mem** - Load register A from memory address
- **LOADB mem** - Load register B from memory address
- **CONB con** - Load a constant value into register B
- **SAVEB mem** - Save register B to memory address
- **SAVEC mem** - Save register C to memory address
- **ADD** - Add A and B and store the result in C
- **SUB** - Subtract A and B and store the result in C
- **MUL** - Multiply A and B and store the result in C
- **DIV** - Divide A and B and store the result in C

Microprocessor/Microcontroller

- **COM** - Compare A and B and store the result in test
- **JUMP addr** - Jump to an address
- **JEQ addr** - Jump, if equal, to address
- **JNEQ** addr - Jump, if not equal, to address
- **JG addr** - Jump, if greater than, to address
- **JGE addr** - Jump, if greater than or equal, to address
- **JL addr** - Jump, if less than, to address
- **JLE addr** - Jump, if less than or equal, to address
- **STOP** - Stop execution

Microprocessor/Microcontroller

- An **opcode** (**op**eration **code**) is the portion of a machine language instruction that specifies the operation to be performed. Their specification and format are laid out in the instruction set architecture of the processor in question (which may be a general CPU or a more specialized processing unit). Apart from the opcode itself, an instruction normally also has one or more specifiers for operands (i.e. data) on which the operation should act, although some operations may have *implicit* operands, or none at all.

Microprocessor/Microcontroller

- Assembly language, or just *assembly*, is a low-level programming language, which uses mnemonics, instructions and operands to represent machine code. This enhances the readability while still giving precise control over the machine instructions.

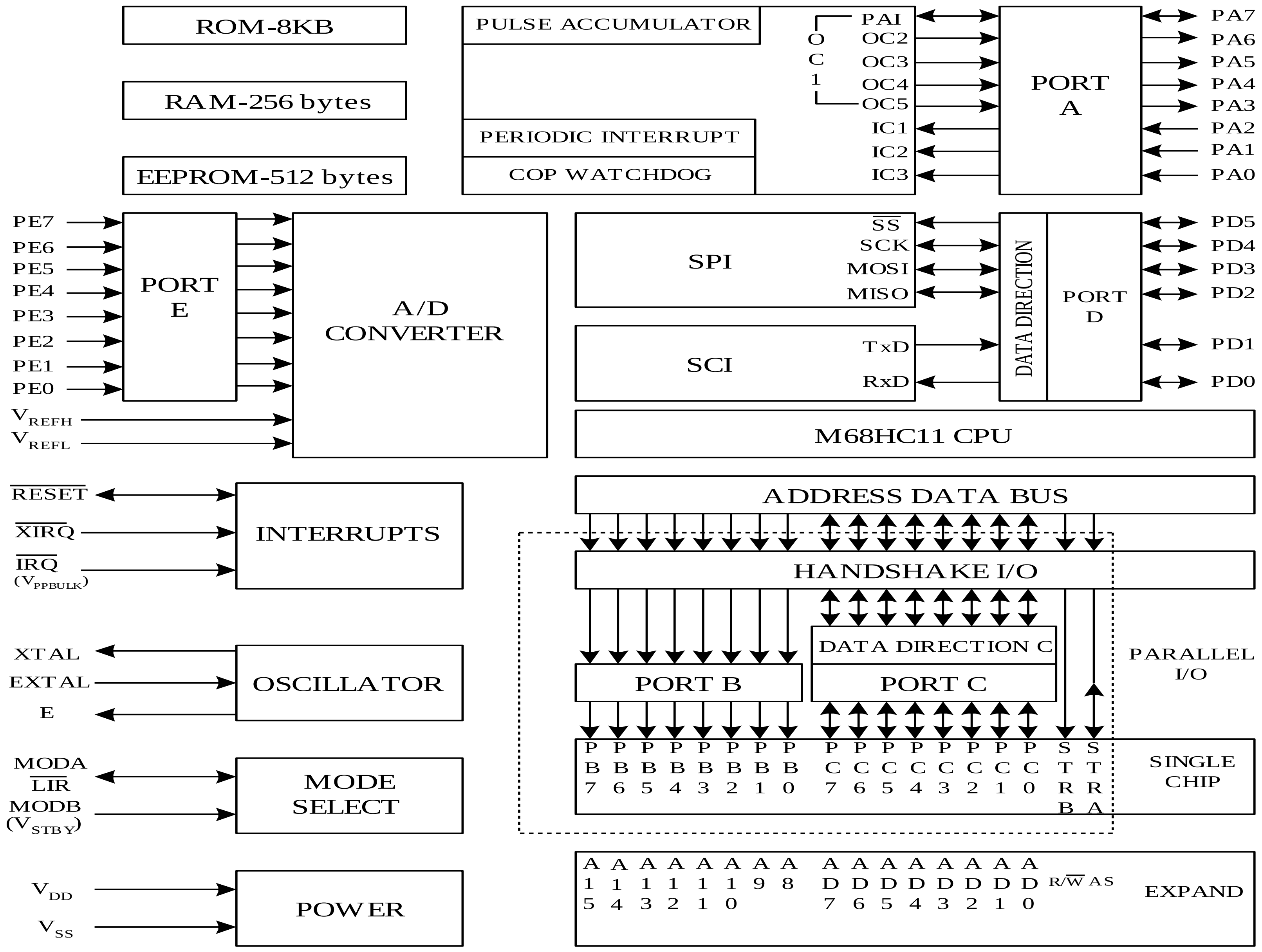


Figure 1.2 68HC11A8 block diagram (redrawn with permission of Motorola)

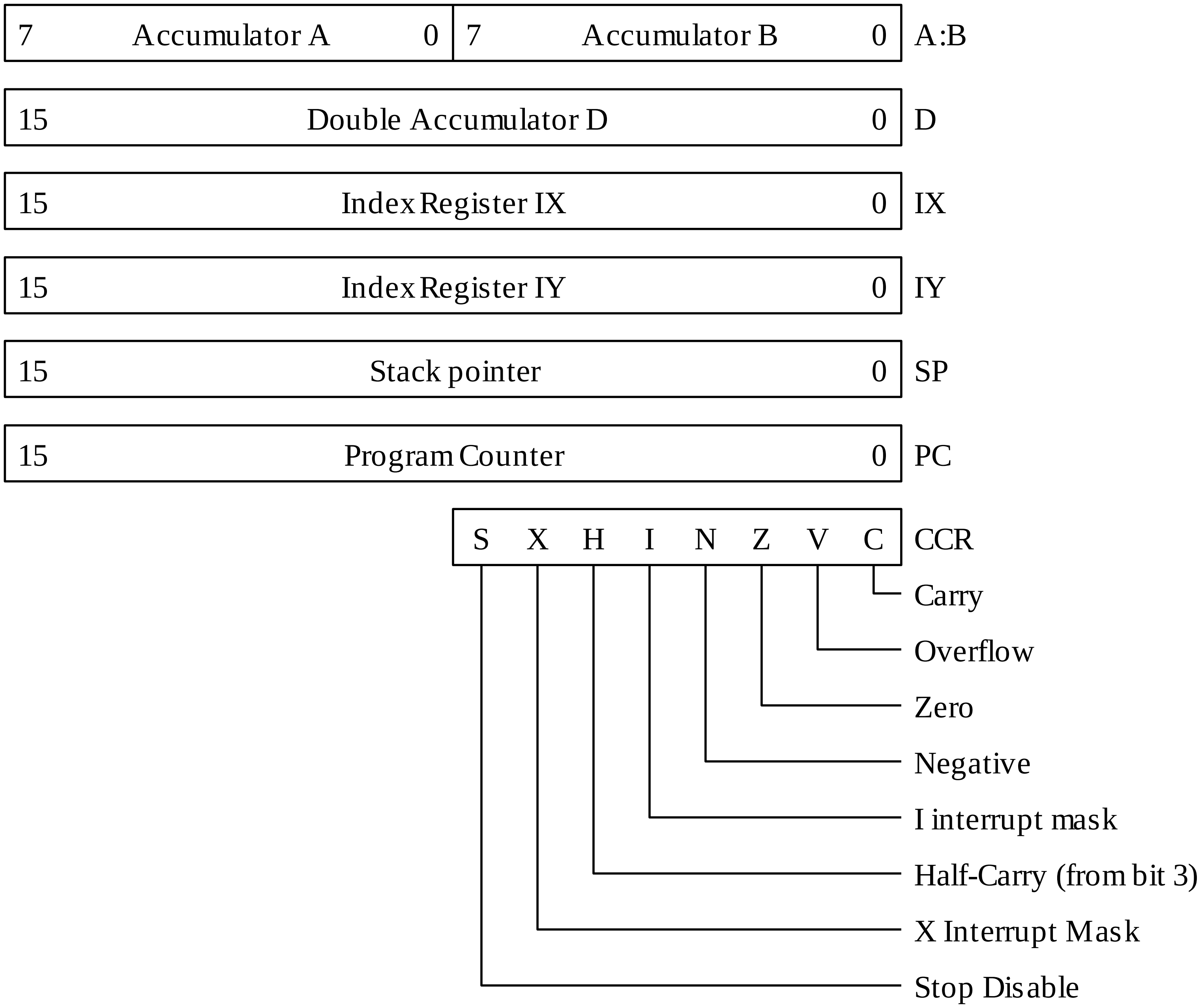
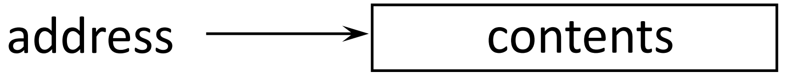


Figure 1.3 MC68HC11 Programmer's model

Microprocessor/Microcontroller

Memory Addressing

Memory consists of addressable locations
A memory location has 2 components: address and contents



Data transfer between CPU and memory involves address bus and data bus

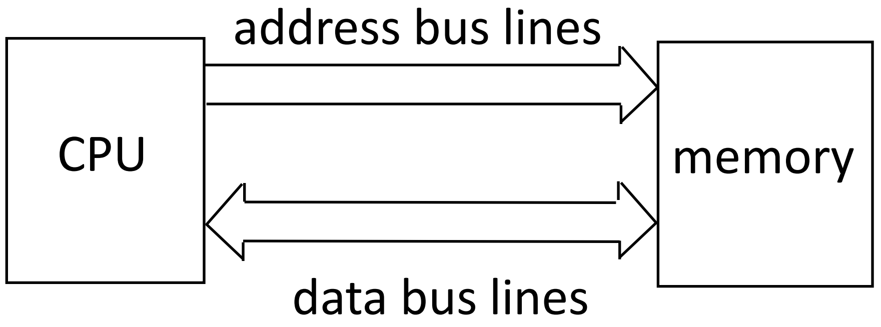


Figure 1.5 Data transfer between CPU and memory

ADDRESSING MODES

Operands needed in an instruction are specified by one of the 6 addressing modes

Immediate mode
Direct mode
Extended mode
Indexed mode
Inherent mode
Relative mode

Microprocessor/Microcontroller

68HC11 addressing modes

Table 1.1 Prefix for number representation

Base	Prefix
binary	%
octal	@
decimal	nothing*
hexadecimal	$

*Note: Some assemblers use &

Immediate mode

The actual operand is contained in the byte or bytes immediately following the instruction opcode

```
LDAA #22
ADDA #@32
LDD #1000
```

Note that the (#) is a critical assembler directive!

Direct mode

A one-byte value is used as the address of a memory operand (located in on-chip SRAM)

```
ADDA  $10
SUBA  $20
LDD  $30
```

Extended mode

A two-byte value is used as the address of a memory operand

```
LDAA  $1000
LDX  $1000
ADDD  $1030
```

Indexed mode

The sum of one of the index registers and an 8-bit value is used as the address of a memory operand

```
ADDA  10,X
LDAA  3,Y
```

Inherent mode

- Operands are implied by the instruction
- No address information is needed

```
      ABA
      INCB
      INX
```

Relative mode

- Used in branch instructions to specify the branch target
- Specified using either a 16-bit value or a label (preferred)

```
      ...
      BEQ  there
      ADDA  #10
      ...
there  DECB
```

A Sample of 68HC11 Instructions

The LOAD instructions

A group of instructions that place a value or copy the contents of a memory location (or locations) into a register

LDAA	<opr>	Load Accumulator A
LDAB	<opr>	Load Accumulator B
LDD	<opr>	Load Double Accumulator D
LDX	<opr>	Load Index Register X
LDY	<opr>	Load Index Register Y
LDS	<opr>	Load Stack Pointer
	<opr>	can be *immediate*, *direct*, *extended*, or *index mode*

Examples

```
LDAA    $10
LDX     #$1000
```

The ADD instruction

A group of instructions perform addition operation

```
ABA
ABX
ABY
ADDA  <opr>
ADDB  <opr>
ADDD  <opr>
ADCA  <opr>
ADCB  <opr>
```

<opr> is specified using *immediate, direct, extended*, or *index mode*

Examples.

```
ADDA #10
ADDA $20
ADDD $30
```

The SUB instruction

A group of instructions that perform the subtract operation

```
SBA
SUBA    <opr>
SUBB    <opr>
SUBD    <opr>
SBCA    <opr>           ; A ← [A] - <opr> - C flag
SBCB    <opr>           ; A ← [B] - <opr> - C flag
        <opr>  can be immediate, direct, extended, or index mode
```

Examples

```
SUBA    #10
SUBA    $10
SUBA    0,X
SUBD    10,X
```

The STORE instruction

A group of instructions that store the contents of a register into a memory location or memory locations

```
STAA    <addr>
STAB    <addr>
STD     <addr>
STX     <addr>
STY     <addr>
STS     <addr>
```

<addr> can be *direct*, *extended*, or *index* mode

Examples:

```
STAA    $20
STAA    10,X
STD     $10
STD     $1000
STD     0,X
```

The 68HC11 Machine Code

A 68HC11 **instruction consists of 1 to 2 bytes of opcode and 0 to 3 bytes of operand information**

Examples

Assembly instruction	Machine instructions (in hex format)
LDAA #29	86 1D
STAA $00	97 00
ADDA $02	9B 02
STAA $01	97 01
INY	18 08

Microprocessor/Microcontroller

machine code	**assembly instruction format**
01	NOP
86	LDAA IMM
96	LDAA DIR
C6	LDAB IMM
D6	LDAB DIR
CC	LDD IMM
DC	LDD DIR
8B	ADDA IMM
9B	ADDA DIR
CB	ADDB IMM
DB	ADDB DIR
C3	ADDD IMM
D3	ADDD DIR
97	STAA DIR
D7	STAB DIR
DD	STD DIR

Microprocessor/Microcontroller

The 68HC11 Instruction Execution Cycle

- Perform a sequence of read cycles to fetch instruction opcode byte and address information.
- Optionally perform read cycle(s) required to fetch the memory operand.
- Perform the operation specified by the opcode.
- Optionally write back the result to a register or a memory location.

- Consider the following 3 instructions

Assembly instruction	*Memory location*	*Opcode*
LDAA $2000	$C000	B6 20 00
ADAA $3000	$C003	BB 30 00
STAA $2000	$C006	B7 20 00

Instruction **LDAA $2000**

Step 1. Place the value in PC on the address bus with a request to read the contents of that location.

Step 2. The opcode byte **$B6** at $C000 is returned to the CPU and PC is incremented by 1.

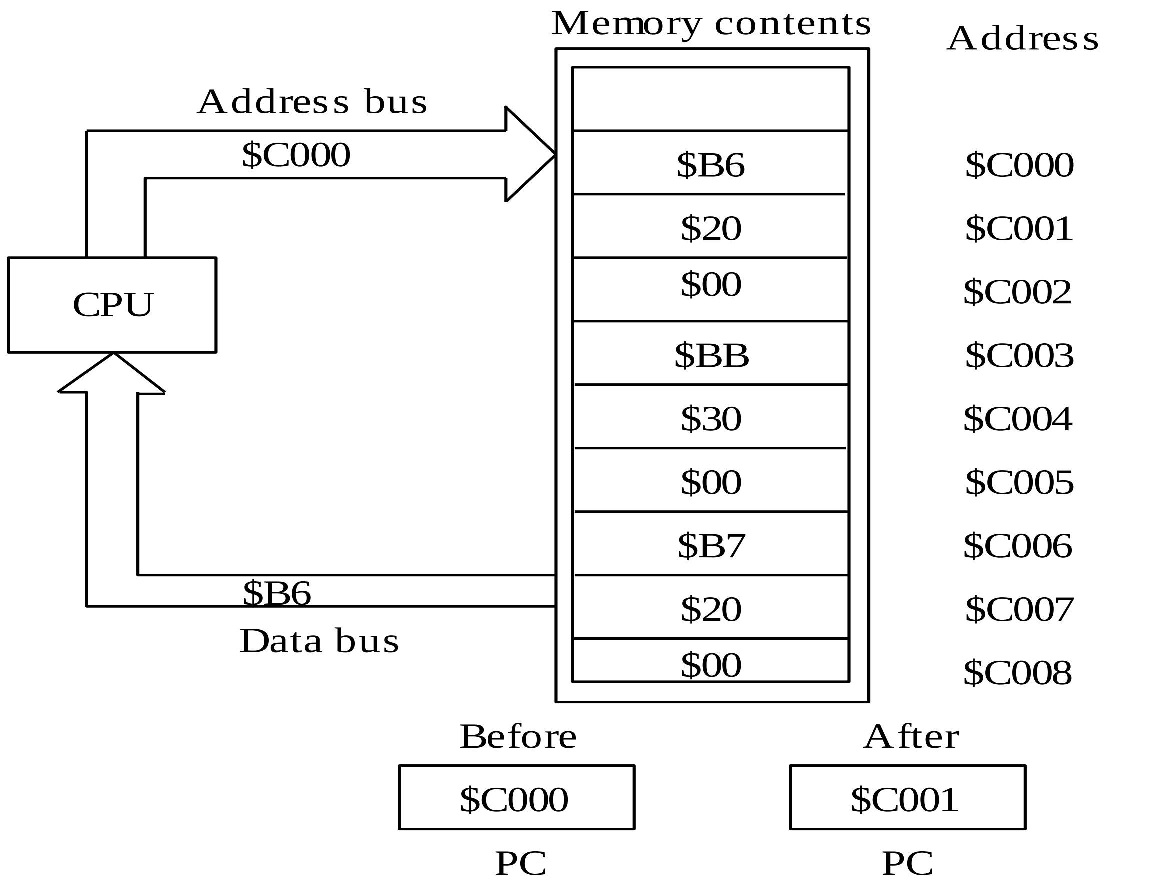


Figure 1.10 Instruction 1--Opcode read cycle

Step 3. CPU performs two read cycles to obtain the extended address $2000 from locations $C001 and $C002. At the end the value of PC is incremented to $C003

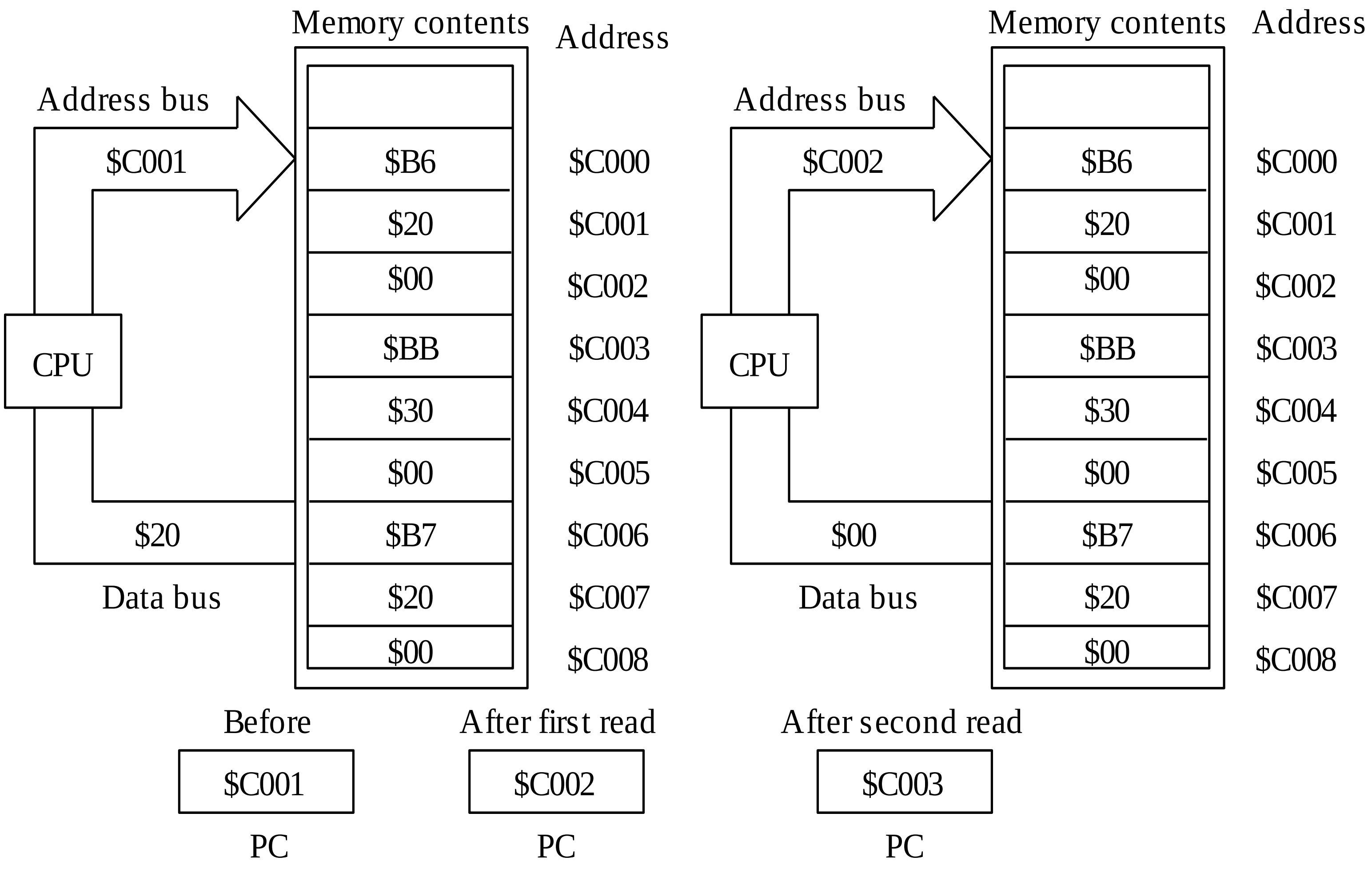


Figure 1.11 Instruction 1--address byte read cycles

Step 4. The CPU performs another read to get the contents of the memory location at $2000, which is $19. The value $19 will be loaded into accumulator A.

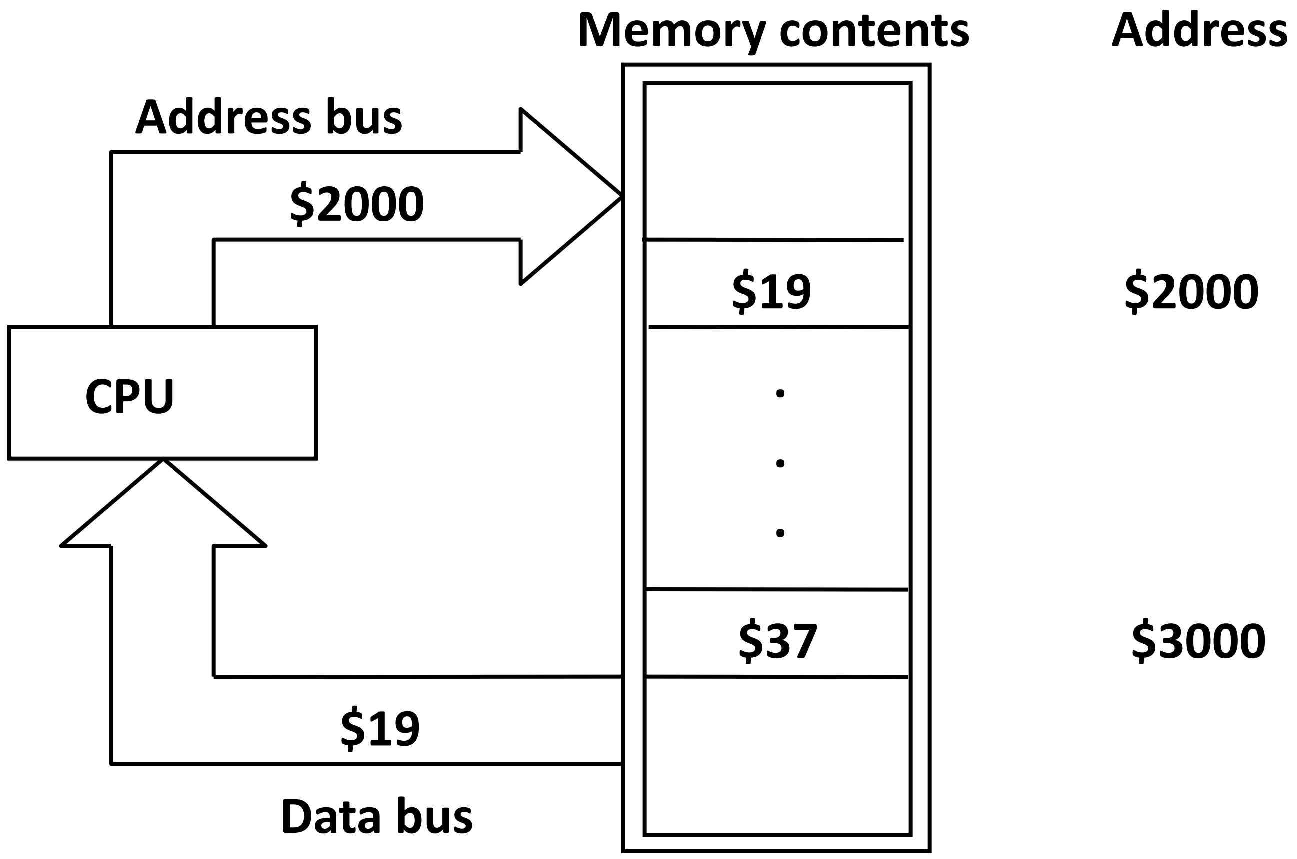


Figure 1.12 Instruction 1--execution read cycle

The End

Microprocessor/Microcontroller