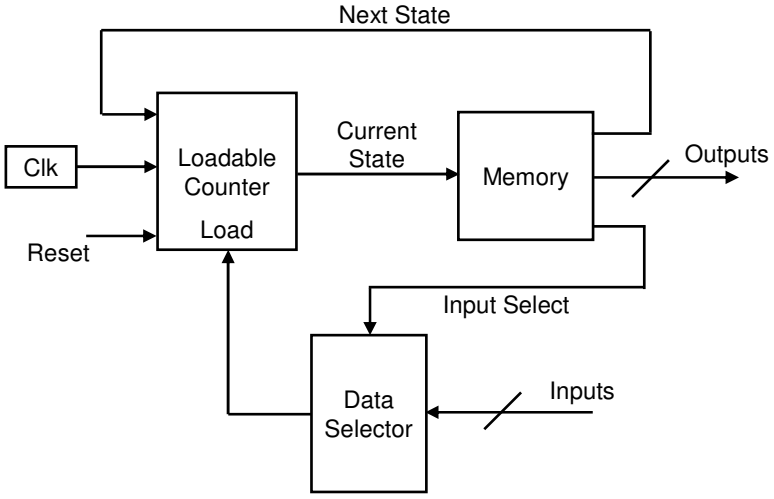
University of British Columbia
Electrical and Computer Engineering
Digital Design and Microcomputers CPEN312

Lecture 12: Introduction to Microcomputers

Dr. Jesús Calviño-Fraga. P.Eng.
Department of Electrical and Computer Engineering, UBC
Office: KAIS 3024
E-mail: jesusc@ece.ubc.ca
Phone: (604)-827-5387

March 2, 2018

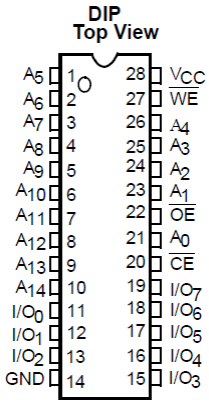
Programmable State Machine



Memory

- Typically, memory has the following signals:
  - An address bus. This allows us to select each particular memory location.
  - A data bus. Here is where we put the information to be stored to the memory, or where we get the information previously stored.
  - A control bus. These are the signals that tell the memory to write, read, or disable/enable the whole memory.

Random Access Memory (RAM)



CY62256N

256K (32K x 8) Static RAM

Address Bus: Signals A_0 to A_{14}. Since we have 15 address signals, the memory has 2^{15}=32768 locations.

Data Bus: Signals I/O_0 to I/O_7. It is an 8-bit data bus.

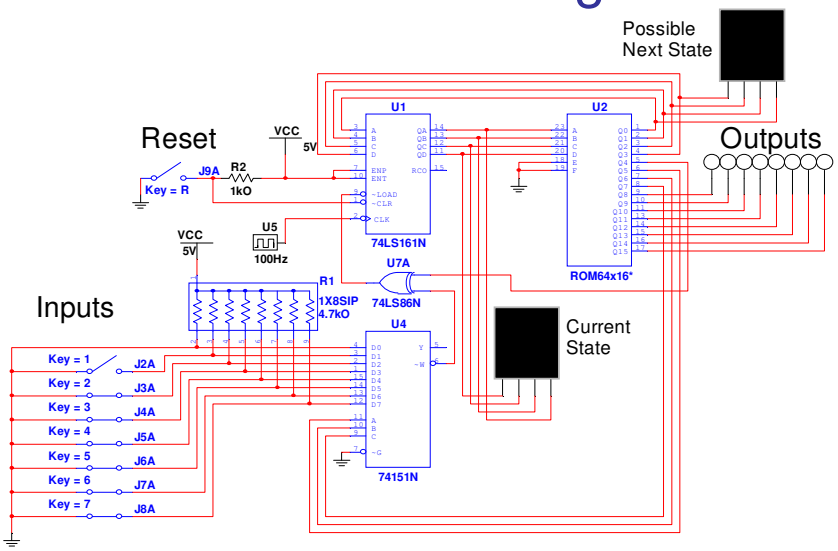
Control Bus: Signals CE*, OE*, and WE*.

CE*: Enables this IC for read or write.

OE*: Makes the data bus an output. Read from memory.

WE*: Makes the data bus an input. Write to memory

PSM: Simulation using Multisim

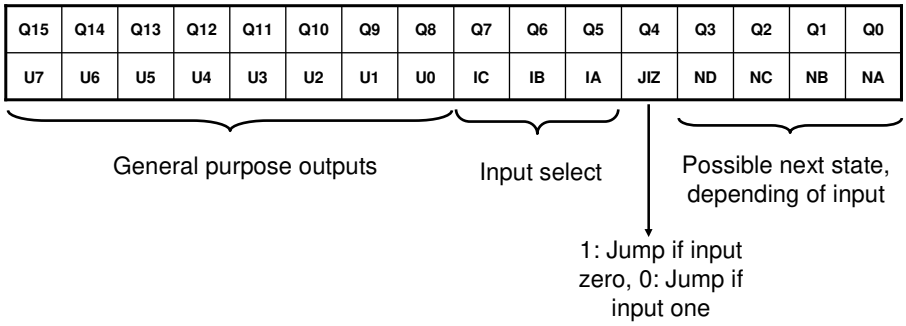


PSM: Simulation using Multisim

- The program is stored in the 64x16 ROM (64 words, 16-bit each).
- To change the program in Multisim, double click the memory, then click “Edit Model”.
- Modify the output of the memory for each location according to your program. Then click “Change part model”, followed by Ok.
- To simulate the Programmable State Machine press F5.

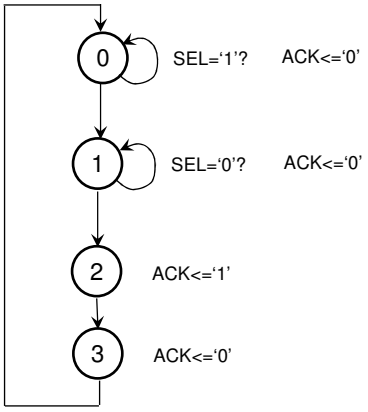
PSM: Operation Code

- Each instruction in our programmable state machine consist of 16-bits with the following meanings:



PSM Example 1

- Program the following state diagram:

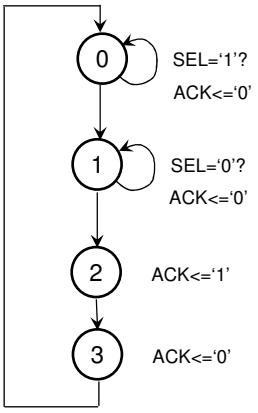


PSM: Signal Assignments

Signal	PSM
'0'	Input 0 (I_A=0, I_B=0, I_C=0)
'SEL'	Input 2 (I_A=0, I_B=1, I_C=0)
'ACK'	Output 0 (U0)

PSM Code

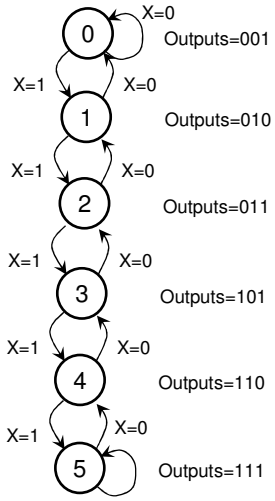
State				Outputs								Input select				Next State			
D	C	B	A	U7	U6	U5	U4	U3	U2	U1	U0	IC	IB	IA	JIZ	ND	NC	NB	NA
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	X	X	X	X
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0



PSM Example 2

- The elevator in a six storey building in Vancouver needs a counter to keep track of its position. As is common in Vancouver there are neither floors ‘0’ nor ‘4’. Write a program for the programmable state machine to show the floor position. Two signals are provided: the clock ‘CLK’ and the direction ‘X’. When X=’1’ the elevator is going up. When X=’0’ the elevator is going down.

Example 2 State Diagram



PSM: Signal Assignments

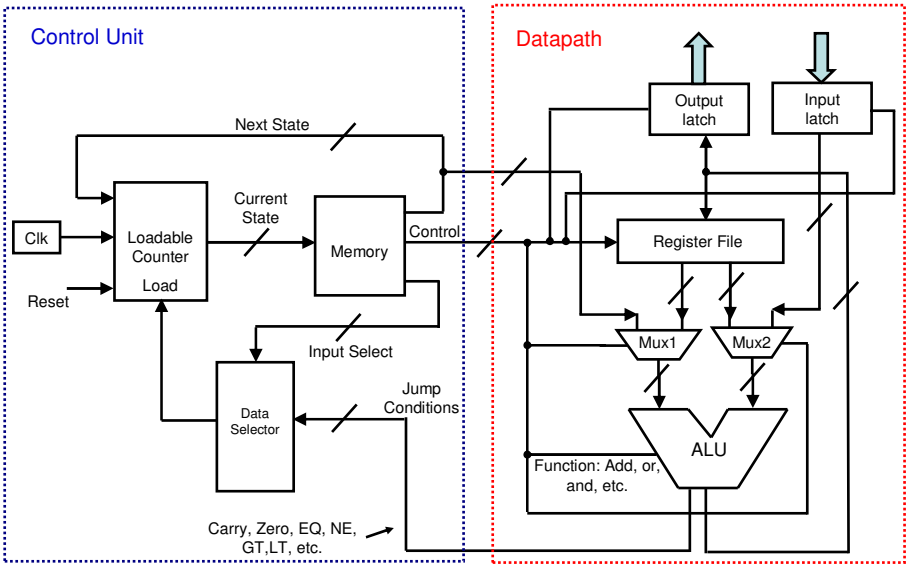
Signal	PSM
'0'	Input 0 (I_A=0, I_B=0, I_C=0)
'X'	Input 2 (I_A=0, I_B=0, I_C=1)
'F0'	Output 0 (U0)
'F1'	Output 1 (U1)
'F2'	Output 2 (U2)

PSM Code

Limitation of PSM: jump to another "instruction" or next "instruction" only!

State				Outputs								Input select				Next State			
D	C	B	A	U7	U6	U5	U4	U3	U2	U1	U0	IC	IB	IA	JIZ	ND	NC	NB	NA
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0
0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0
0	0	1	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	1
0	0	1	1	0	0	0	0	0	1	0	1	0	0	1	1	0	0	1	0
0	1	0	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	1	1
0	1	0	1	0	0	0	0	0	1	1	1	0	0	1	1	0	1	0	0
0	1	1	0	0	0	0	0	0	1	1	1	0	0	0	1	0	1	0	1

A "Simple" Central Processing Unit (with I/O)



Central Processing Unit (CPU)

- Digital Circuit that performs:
  - Logic
  - Arithmetic
  - Control
  - Input/Output
- CPU includes:
  - Control Unit (PSM)
  - ALU (Made with logic)
  - Registers (or memory, made with flip flops)

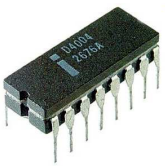
What can we do with the "Simple" CPU

- Logic: AND, OR, NOT, XOR, etc. ANY ARBITRARY LOGIC WE WANT!
- Arithmetic: ADD, SUB, INC, DEC.
- Control: JUMP always, JUMP if Carry, JUMP if Zero, JUMP if EQ, JUMP if NEQ, JUMP if GT, JUMP if LT, etc. Jump to whatever the ALU provides!
- Copy registers. R2=R1+0
- Write to External port or Read from External port.
- Load or operate with immediate values.
- Many designs include a latch connected to the output of the ALU. That "register" is called the **Accumulator**.

CPU: design vs. buy

- Before 1971 (when the first microprocessor was introduced) it was very common to 'design' instead of 'buy'.
- Today even powerful CPUs are very inexpensive.
- When you 'buy', compilers, documentation, and examples are available. When you 'design' you have to develop also compilers, examples, and write documentation (ugh!)
- Therefore we are going to 'buy' instead of 'design'.

First Microprocessor (?)



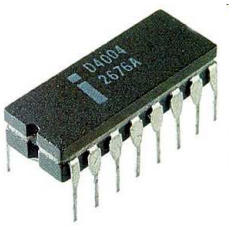
The designers of the 4004 were Federico Faggin and Ted Hoff of Intel, and Masatoshi Shima of Busicom.

First delivery around March 1971

A Busicom made calculator

Concepts: what is a microcomputer?

- A microcomputer is a computer built using a **microprocessor.**
- A **microprocessor** is a single integrated circuit (IC) which incorporates the functions of a central processing unit (CPU).



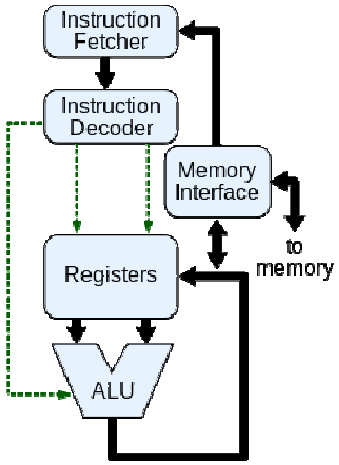
The Intel 4004

Atmel AT91SAM7XC512

Silicon Labs C8051F330

Simple CPU

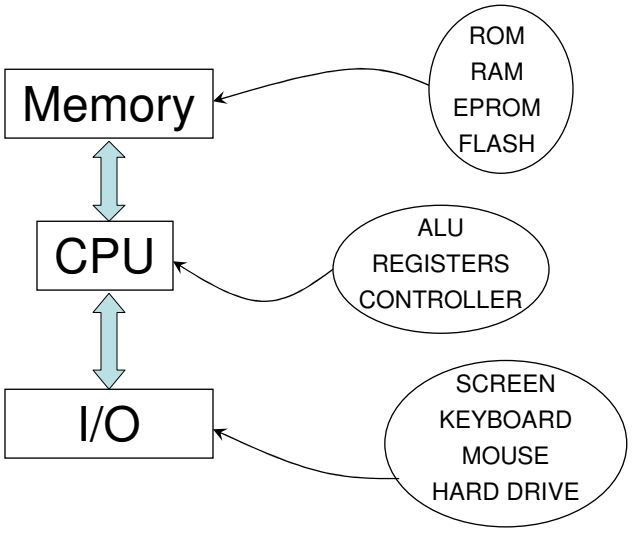
https://en.wikipedia.org/wiki/File:CPU_block_diagram.svg



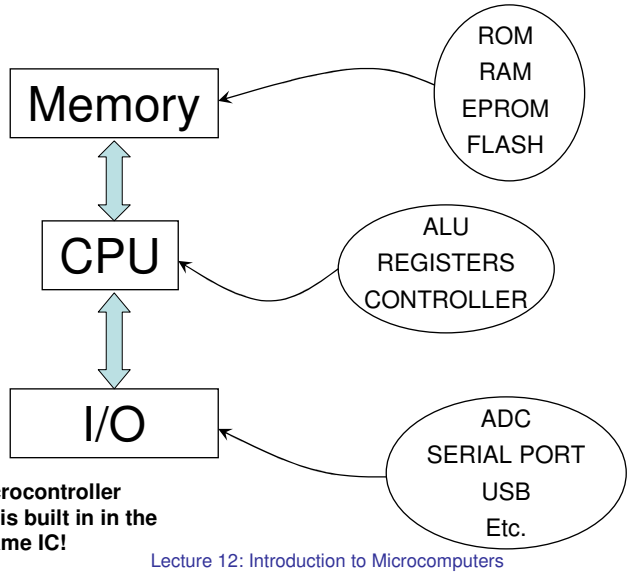
Computers, microprocessor, or microcontroller?

- A computer can be defined as anything that computes. It is commonly associated with some electro/mechanical device.
- A microprocessor is an integrated circuit Central Processor Unit (CPU).
- A microcontroller is a microprocessor + memory + I/O! All wrapped in to one single integrated circuit. A microcontroller is a complete microcomputer in an IC!

Electronic Digital Computer



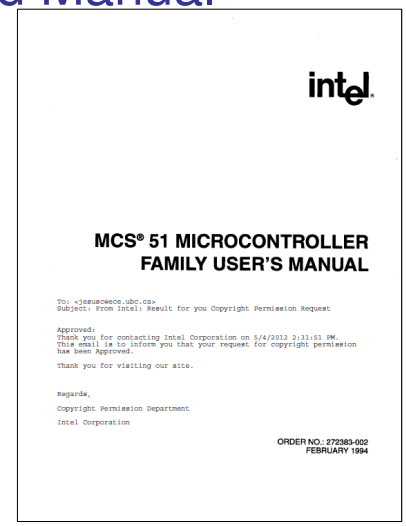
Microcontroller



In a Microcontroller everything is built in in the same IC!

Second Part of the Course: Required Manual

- The MCS-51 Microcontroller user's manual will be our reference for the second half of the course. **Chapters 1, 2, and 3 only!**
- The final exam will be open book.
  - Sharing of material will not be allowed.
  - Printed material only. Electronic devices will not be allowed.
- Available on Connect


intel

MCS® 51 MICROCONTROLLER FAMILY USER'S MANUAL

To: <jesuscweee.ubc.ca>
Subject: From Intel: Result for you Copyright Permission Request

Approved:
Thank you for contacting Intel Corporation on 5/4/2012 2:31:51 PM.
This email is to inform you that your request for copyright permission has been Approved.

Thank you for visiting our site.

Regards,

Copyright Permission Department

Intel Corporation

ORDER NO.: 272383-002
FEBRUARY 1994

Why I selected the 8051 (AKA MCS 51) processor for this course?

- It has been around since 1980.
- Dozens of companies manufacture it.
- Lots of resources available: books, source code, tutorials, debuggers, compilers, etc.
- New and improved versions come out every year.
- Lots of VHDL/Verilog implementations available, both free and commercial.
- Reasonable number of instructions for an introductory course:

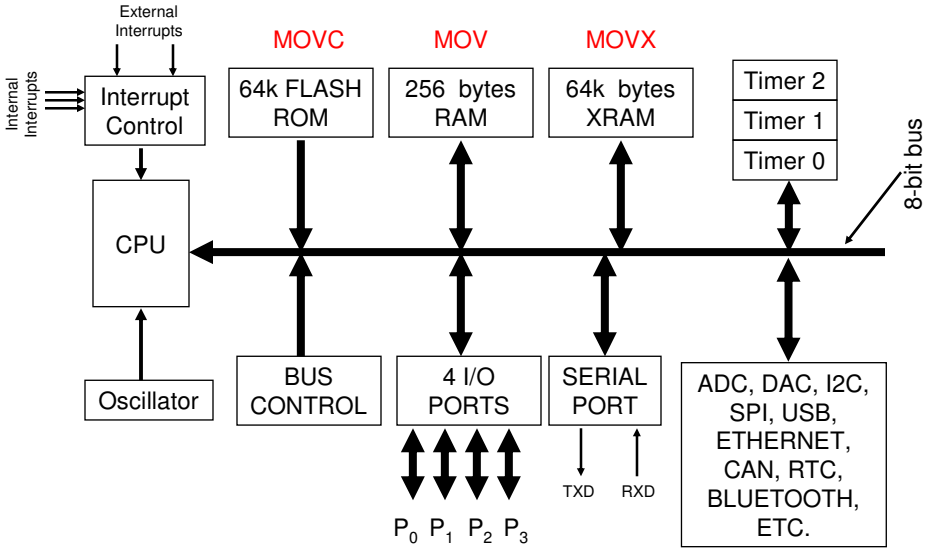
Approximate number of Instructions for some processors (last time I checked!)

Processor	Instructions
Pentium 4	360
ARM	148
NIOS II	86
8051	57
PIC 18	33

VHDL 8051 for CPEN312

- Basic 8051 downloaded from http://opencores.org (T51 project) and added support and features for the Altera's DE0-CV board such as a boot loader, USB, debugger, as well as all the switches and displays/LEDs. Also fixed some bugs!
- Available from the course web page. There is also a guide to help you get started with CV-8052 soft processor.

Modern 8051 Block Diagram



Memory organization

- von Neumann: Code and data share the same memory space. Most desktop/laptop computers are like this.
- Harvard: Code and data use different memory spaces. Many microcontrollers are like this. The 8051 uses a Harvard architecture.

Programming a Microcontroller: Assembly Language

- ALL microprocessors can be programmed in Assembly Language.
- Assembly is a low-level programming language:
  - Usually, there are only very simple instructions available. Check for example what can you do with the "Simple" CPU presented earlier.
  - In some architectures there are only a few instructions available.
- Assembly language uses mnemonics to represent the low-level machine operations. These include logic, arithmetic, transfer, I/O, branching instructions (jumps), etc.

Using a C compiler in your personal computer

```
#include <stdio.h>
#include <stdlib.h>
void main (void)
{
    printf("Hello, world!\n");
}
```

```
C:\Courses\CPEN312\Source>gcc -S Hello_PC.c
```

The '–S' switch tells the compiler to save the assembly code for the passed 'c' program:

Assembly from C for PC

```
        .file   "Hello_PC.c"
        .def    ___main;        .scl    2;
        .type   32;     .endef
        .section .rdata,"dr"
LC0:
        .ascii "Hello, world!\0"
        .text
.globl _main
        .def    _main;  .scl    2;      .type
        32;     .endef
_main:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp
        call    ___main
        leaq    LC0(%rip), %rcx
        call    _puts
        leave
        ret
        .def    _puts;  .scl    2;      .type
        32;     .endef
```


Using a C compiler for the 8051

```
#include<stdio.h>
#include<stdlib.h>

#define CLK 33333333L
#define BAUD 115200L
#define TIMER_2_RELOAD (0x10000L-(CLK/(32L*BAUD)))

void main (void)
{
    setbaud_timer2(TIMER_2_RELOAD);
    printf("Hello, world!\n");
}
```

Assembly from C for 8051

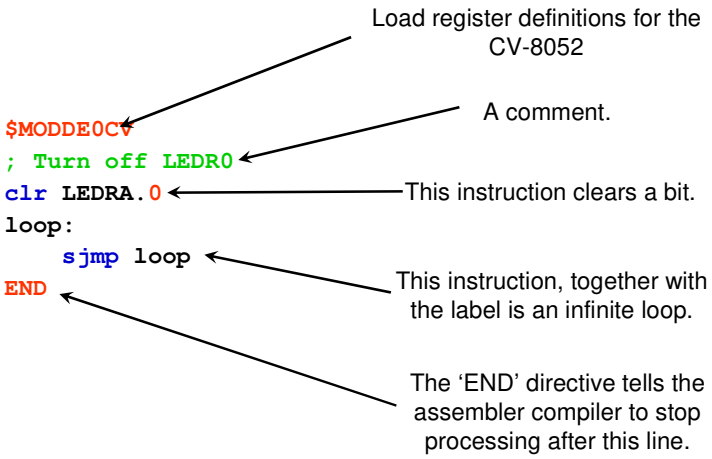
```
_main:
        using   0
        mov     dptr,#0xFFF7
        lcall   _setbaud_timer2
        mov     a,#__str_0
        push    acc
        mov     a,#(__str_0 >> 8)
        push    acc
        mov     a,#0x80
        push    acc
        lcall   _printf
        dec     sp
        dec     sp
        dec     sp
        ret

        rseg R_CONST
__str_0:
        db 'Hello, world!'
        db 0x0A
        db 0x00
end
```


Why is it important to learn Assembly Language programming?

- Assembly language is the link between software and hardware. It is the closest to machine language (the state machine or controller) you want to be.
- Assembly programs can be extremely efficient both in speed and code size.
- Assembly programs allows you to access 'hidden' hardware features.
- Sometimes the only compiler available is an assembler.
- Assembly language lets you see what the high level compiler does: sometimes crucially important!

First Assembly Program



8051 Instruction Set

1. Arithmetic Operations
2. Logical Operations
3. Data Transfer
4. Boolean Variable Manipulation
5. Program Branching and Machine Control

The cycles per instruction for the tables that follow are for the CV_8052

Arithmetic Operations (ADD)

Mnemonic	Opcode	B	C	Function
ADD A, Rn	28H-2FH	1	1	(A) = (A) + (Rn)
ADD A, direct	25H	2	2	(A) = (A) + (direct)
ADD A, @Ri	26H-27H	1	1	(A) = (A) + ((Ri))
ADD A, #data	24H	2	2	(A) = (A) + #data
ADDC A, Rn	38H-3FH	1	1	(A) = (A) + (C) + (Rn)
ADDC A, direct	35H	2	2	(A) = (A) + (C) + (direct)
ADDC A, @Ri	36H-37H	1	1	(A) = (A) + (C) + ((Ri))
ADDC A,#data	34H	2	2	(A) = (A) + (C) + #data

Arithmetic Operations (SUB)

Mnemonic	Opcode	B	C	Function
SUBB A, Rn	98H-9FH	1	1	(A) = (A) - (C) - (Rn)
SUBB A, direct	95H	2	2	(A) = (A) - (C) - (direct)
SUBB A, @Ri	96H-97H	1	1	(A) = (A) - (C) - ((Ri))
SUBB A, #data	94H	2	2	(A) = (A) - (C) - #data

Arithmetic Operations (INC)

Mnemonic	Opcode	B	C	Function
INC A	04H	1	1	(A) = (A) + 1
INC Rn	08H-0FH	1	1	(Rn) = (Rn) + 1
INC direct	05H	2	2	(direct) = (direct) + 1
INC @Ri	06H-07H	1	1	((Ri)) = ((Ri)) + 1
INC DPTR	A3H	1	1	(DPTR) = (DPTR) + 1

Arithmetic Operations (DEC)

Mnemonic	Opcode	B	C	Function
DEC A	14H	1	1	(A) = (A) - 1
DEC Rn	18H-1FH	1	1	(Rn) = (Rn) - 1
DEC direct	15H	2	2	(direct) = (direct) -1
DEC @Ri	16H-17H	1	1	((Ri)) = ((Ri)) - 1

Arithmetic Operations (Other)

Mnemonic	Opcode	B	C	Function
MUL AB	A4H	1	1	(B15–8), (A7–0) = (A) x (B)
DIV AB	84H	1	10	(B15–8), (A7–0) = (A) / (B)
DA A	D4H	1	1	Converts (A) to BCD

Logic Operations (AND)

Mnemonic	Opcode	B	C	Function
ANL A, Rn	58H-5FH	1	1	(A) = (A) AND (Rn)
ANL A, direct	55H	2	2	(A) = (A) AND (direct)
ANL A, @Ri	56H-57H	1	1	(A) = (A) AND ((Ri))
ANL A, #data	54H	2	2	(A) = (A) AND #data
ANL direct, A	52H	2	2	(direct) = (direct) AND A
ANL direct, #data	53H	3	3	(direct) = (direct) AND #data

Logic Operations (OR)

Mnemonic	Opcode	B	C	Function
ORL A, Rn	48H-4FH	1	1	(A) = (A) OR (Rn)
ORL A, direct	45H	2	2	(A) = (A) OR (direct)
ORL A, @Ri	46H-47H	1	1	(A) = (A) OR ((Ri))
ORL A, #data	44H	2	2	(A) = (A) OR #data
ORL direct, A	42H	2	2	(direct) = (direct) OR (A)
ORL direct, #data	43H	3	3	(direct) = (direct) OR #data

Logic Operations (XOR)

Mnemonic	Opcode	B	C	Function
XRL A, Rn	68H-6FH	1	1	(A) = (A) XOR (Rn)
XRL A, direct	65H	2	2	(A) = (A) XOR (direct)
XRL A, @ Ri	66H-67H	1	1	(A) = (A) XOR ((Ri))
XRL A, #data	64H	2	2	(direct) = (A) XOR #data
XRL direct, A	62H	2	2	(direct) = (direct) XOR (A)
XRL direct, #data	63H	3	3	(direct) = (direct) XOR #data

Logic Operations (Other)

Mnemonic	Opcode	B	C	Function
CLR A	E4H	1	1	(A) = 0
CPL A	F4H	1	1	(A) = (A')
RL A	23H	1	1	Rotate (A) Left
RLC A	33H	1	1	Rotate (A) Left Through Carry
RR A	03H	1	1	Rotate (A) Right
RRC A	13H	1	1	Rotate (A) Right Through Carry
SWAP A	C4H	1	1	(A3-0) ↔ (A7-4)

Data Transfer (MOV 1 of 3)

Mnemonic	Opcode	B	C	Function
MOV A, Rn	E8H-EFH	1	1	(A) = (Rn)
MOV A, direct	E5H	2	2	(A) = (direct)
MOV A, @Ri	E6H-E7H	1	1	(A) = ((Ri))
MOV A, #data	74H	2	2	(A) = #data
MOV Rn, A	F8H-FFH	1	1	(Rn) = (A)
MOV Rn, direct	A8H-AFH	2	2	(Rn) = (direct)
MOV Rn, #data	78H-7FH	2	2	(Rn) = #data
MOV direct, A	F5H	2	2	(direct) = (A)
MOV direct, Rn	88H-8FH	2	2	(direct) = (Rn)

Data Transfer (MOV 2 of 3)

Mnemonic	Opcode	B	C	Function
MOV direct1, direct2	85H	3	3	(direct1) = (direct2) (source) (destination)
MOV direct, @Ri	86H-87H	2	2	(direct) = ((Ri))
MOV direct, #data	75H	3	3	(direct) = #data
MOV @Ri, A	F6H-F7H	1	1	((Ri)) = A
MOV @Ri, direct	A6H-A7H	2	2	((Ri)) = (direct)
MOV @Ri, #data	76H-77H	2	2	((Ri)) = #data
MOV DPTR, #data16	90H	3	3	(DPTR) = #data15-0 (DPH) = #data15-8 (DPL) = #data7-0
MOVC A, @A + DPTR	93H	1	4	(A) = ((A) + (DPTR))

Data Transfer (MOV 3 of 3)

Mnemonic	Opcode	B	C	Function
MOVC A, @A + PC	83H	1	4	(A) = ((A) + (PC))
MOVX A, @Ri	E2H-E3H	1	2	(A) = ((Ri))
MOVX A, @DPTR	E0H	1	2	(A) = ((DPTR))
MOVX @Ri, A	F2H-F3H	1	1	((Ri)) = (A)
MOVX @DPTR,A	F0H	1	1	((DPTR)) = (A)

Data Transfer (Other)

Mnemonic	Opcode	B	C	Function
PUSH direct	C0H	2	3	(SP) = (SP) + 1 ((SP)) = (direct)
POP direct	D0H	2	3	(direct) = ((SP)) (SP) = (SP) - 1
XCHA, Rn	C8H-CFH	1	1	(A) = (Rn)
XCHA, direct	C5H	2	2	(A) = (direct)
XCHA, @Ri	C6H-C7H	1	1	(A) = ((Ri))
XCHD A, @Ri	D6H-D7H	1	1	(A3-0) = ((Ri3-0))

Boolean Variable Manipulation

Mnemonic	Opcode	B	C	Function
CLR C	C3H	1	1	(C) = 0
CLR bit	C2H	2	2	(bit) = 0
SETB C	D3H	1	1	(C) = 1
SETB bit	D2H	2	2	(bit) = 1
CPL C	B3H	1	1	(C) = (C)
CPL bit	B2H	2	2	(bit) = (bit)
ANL C, bit	82H	2	2	(C) = (C) AND (bit)
ANL C, bit’	B0H	2	2	(C) = (C) AND (bit’)
ORL C, bit	72H	2	2	(C) = (C) OR (bit)
ORL C, bit’	A0H	2	2	(C) = (C) OR (bit’)
MOV C, bit	A2H	2	2	(C) = (bit)
MOV bit, C	92H	2	2	(bit) = (C)

Program Branching (1 of 3)

Mnemonic	Opcode	B	C	Function
ACALL addr 11	AAA10001	2	3	(PC) = (PC) + 2 (PC10-0) = page address
LCALL addr 16	12H	3	3	(PC) = addr15-0
RET	22H	1	3	Return from ACALL/CALL
RETI	32H	1	3	Return from interrupt
AJMP addr 11	AAA00001	2	3	(PC) = (PC) + 2 (PC10-0) = page address
LJMP addr 16	02H	3	3	(PC) = addr15-0
SJMP rel	80H	2	3	(PC) = (PC) + 2 (PC) = (PC) + rel
JMP @A + DPTR	73H	1	2	(PC) = (A) + (DPTR)

Program Branching (2 of 3)

Mnemonic	Opcode	B	C	Function
JZ rel	60H	2	2/3	(PC) = (PC) + 2 IF (A) = 0 THEN (PC) = (PC) + rel
JNZ rel	70H	2	2/3	(PC) = (PC) + 2 IF (A) ≠ 0 THEN (PC) = (PC) + rel
JC rel	40H	2	2/3	(PC) = (PC) + 2 IF (C) = 1 THEN (PC) = (PC) + rel
JNC rel	50H	2	2/3	(PC) = (PC) + 2 IF (C) ≠0 THEN (PC) = (PC) + rel
JB bit, rel	20H	3	3/4	(PC) = (PC) + 3 IF (bit) = 1 THEN (PC) = (PC) + rel
JNB bit, rel	30H	3	3/4	(PC) = (PC) + 3 IF (bit) = 0 THEN (PC) = (PC) + rel
JBC bit, rel	10H	3	3/4	(PC) = (PC) + 3 IF (bit) = 1 THEN (bit) = 0 and (PC) = (PC) + rel

Program Branching (3 of 3)

Mnemonic	Opcode	B	C	Function
CJNE A, direct, rel	B5H	3	3/4	Compare and jump if not equal rel
CJNE A, #data, rel	B4H	3	3/4	Compare and jump if not equal rel
CJNE Rn, #data, rel	B8H-BFH	3	3/4	Compare and jump if not equal rel
CJNE @Ri, #data, rel	B6H-B7H	3	3/4	Compare and jump if not equal rel
DJNZ Rn, rel	D8H-DFH	3	2/3	Decrement and Jump if not zero
DJNZ direct,rel	D5H	3	3/4	Decrement and Jump if not zero
NOP	00H	1	1	(PC) = (PC) + 1