Sensors	Data Acquisition	Memory	Processor	Output	Actuators
Voltage, Current	A/D	RAM	Micro-processor	D/A	Loud-speaker
Temp-erature	Codec	Dual Port RAM	Micro-controller	Digital	Relay, Led Lamp
Light, Pressure, Noise	Digital (Fifo, Latch Register)	Multi Port RAM	DSP	Display	Switch
Audio, Ultrasound		Shared RAM	Transputer	SSI/SPI	Motor
Position, Velocity Acceleration		Fifo	Core-Processor	CAN, Profibus	....
Human Interfaces			DSP Multi-Proc essor System	Network (Ethernet)	
....				Bus-Interfaces	

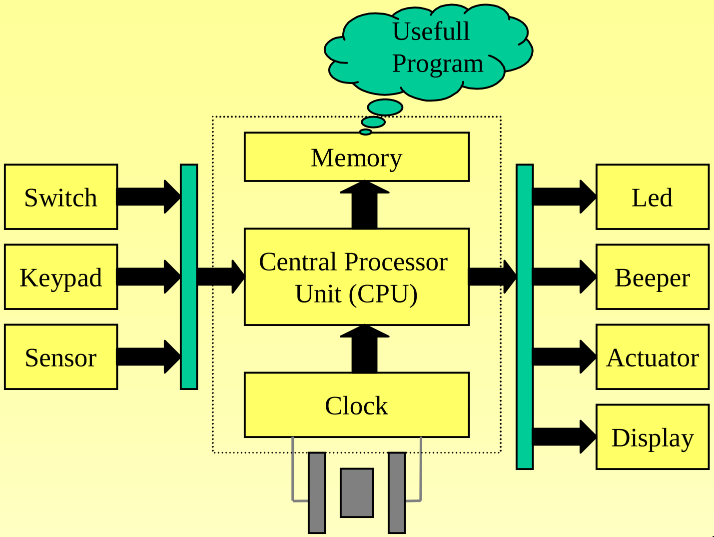
A more generall view to different functional blocks used in embedded systems.

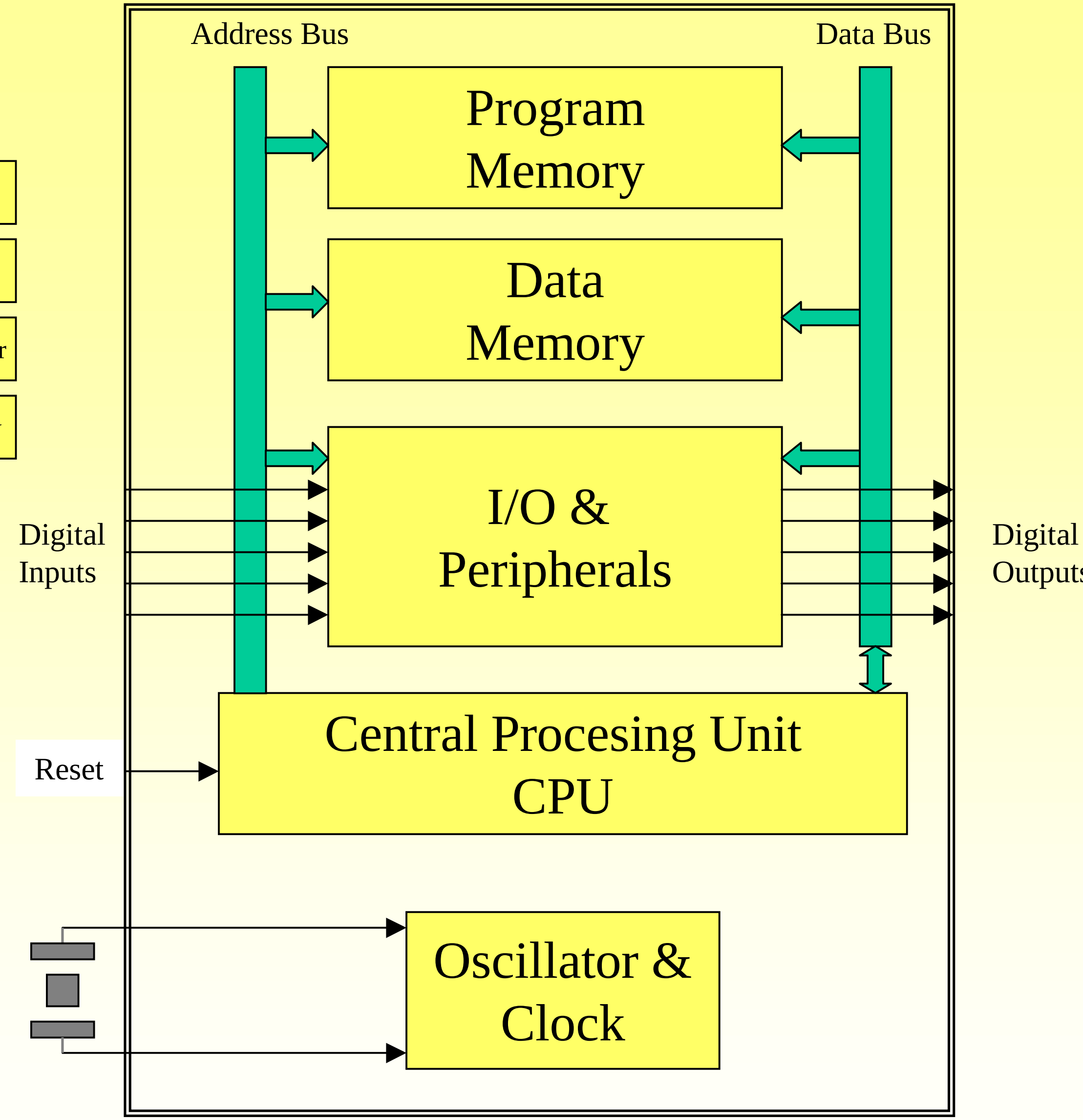
31.10.02

Part of next lectures

Microcontroller (CISC or RISC)

- CISC (Complex Instruction Set Computer)
  - 50 to 250 complex multi-cycle instructions
  - Example: MC68HC705 (Motorola)
- RISC (Reduced Instruction Set Computer)
  - 30 to 50 special one cycle instructions within a pipeline structure
  - Example: PIC 54C5x (MicroChip)



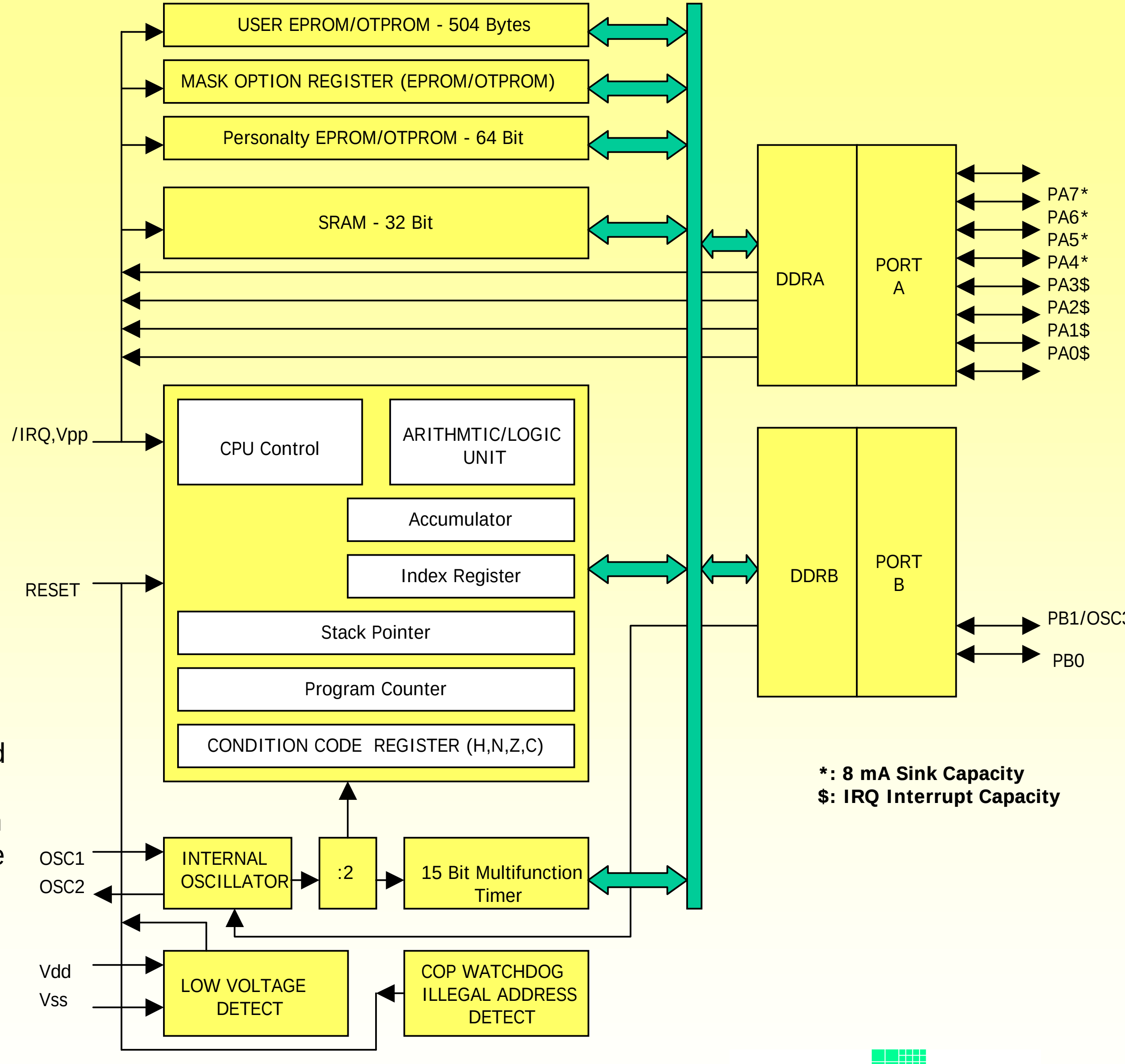


Expanded View of a Microcontroller

Features

MC68HC705K1

- Memory Mapped I/O
- 504 Bytes Erasable ROM
- 32 Bytes User Ram
- 64 Bit Personaltity Rom
- 10 Bidrectional I/O (SPullDown)
- 8 mA Sink Capability on 4 I/O
- External Interrupt on 4 I/O
- Hardware Mask and Flag for External Interrupt
- Fully Static Operation with no Minimum Clock Speed
- On Chip Oscillator (Crystal, Ceramic)
- Computer Operating Properly COP Watchdog
- 15 Bit Multifunction Timing (IRQ)
- Power-Saving Stop, Wait, Halt, and Data-Retention Modes
- 8 x 8 Unsigned Multiply Instruction
- 16 Pin Plastic Dual In-Line Package (PDIP)
- 16 Pin Small Outline Integrated Circuit (SOIC)
- 16 Pin Ceramic (Cerdip)



***: 8 mA Sink Capacity**
$: IRQ Interrupt Capacity

Memory Support

Memory

- The CPU is able to address 1 Kbyte of memory space. Typically the program counter (PC) advances one address through the memory, reading the programm instructions and data.
- The program instructions are hold in the EEPROM part of memory as well as fixed data, user defined vectors, and interrupt service routines.
- The RAM portion of the memory contains the variable data. I/O register are memory-mapped so that the CPU can access their locations in the same way that it accesses all other memory locations.

Input/Output Section

- The first 32 addresses of the memory locations, $0000 to $001F, are the I/O section. There are the addresses of the I/O control registers, status registers, and data registers.

RAM

- Addresses $00E0 to $00FF serve as both the user RAM and the stack RAM. The CPU uses five RAM bytes to store/save the content of all CPU registers before processing an interrupt. If a subroutine is executed, the CPU uses two bytes to store the return address. The stack pointer (SP) decrements during pushes and increments during pulls.

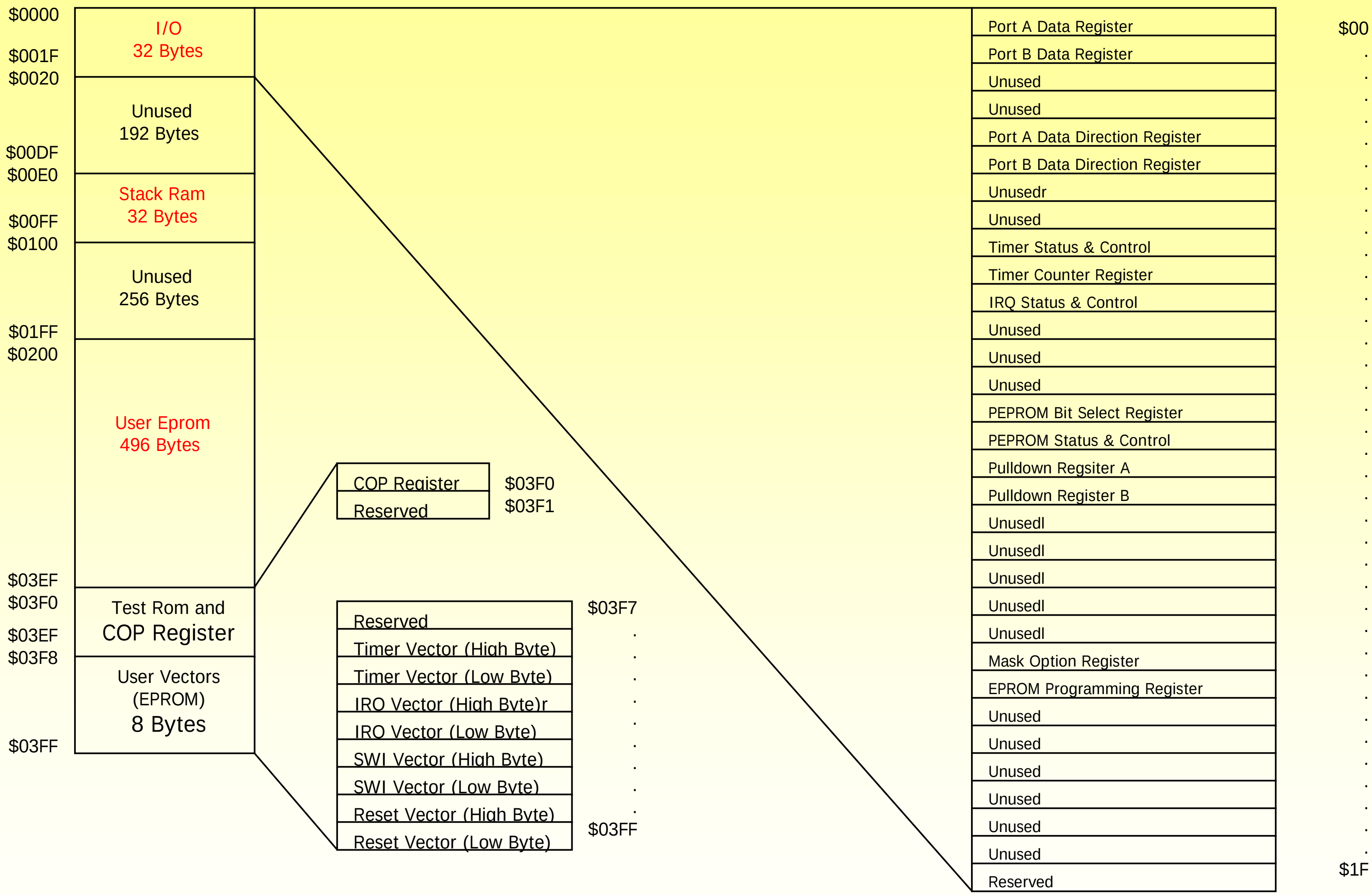
EPROM/OTPROM

- A MCU with a quarz window has 504 bytes of erasable, programmable EPROM (erased by ultravioletlight). A MCU without a quarz window allows only a one-time programming action (OTPROM).

Personality EPROM/OTPROM

- A MCU with a quarz window has an additional 64-bit array of erasable, programmable EPROM. This locations serves as a personality memory area. Without a quarz window, this area is a one time programmable memory location.

A memory map is a pictorial representation of the total MCU memory space.



Typical Memory Map

CPU Registers: In the M68HC05 there are only 5 registers, because it is a relatively simple CPU.

Accumulator:

- General purpose 8-bit register
- The MCU uses the accumulator to hold operands and results of arithmetic/nonarithmetic results.

Index Register

- In the indexed addressing modes, the MCU uses the byte in the index register to determine the conditional address of the operand; the 8-bit index register generally serve as an additional temporary location

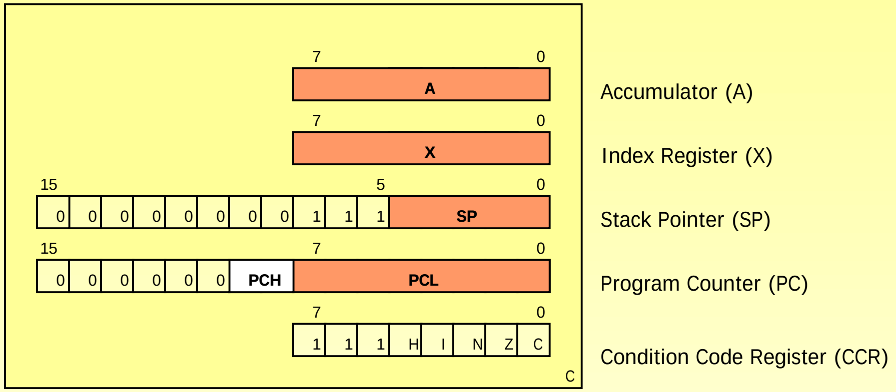
Stack Pointer

- A 16-bit register that contains the address of the next location on stack. If a reset occurs the SP is set to $00FF. The address in the stack pointer decrements as data is pushed onto the stack and increments as data is pulled from the stack. The eleven MSBs are fixed; the stack handles addresses from $00E0 to $00FF. A subroutine has to be limited to 32 locations (only in this example). A subroutine uses two stack locations; an interrupt uses five locations.

Condition Code Register

- The CCR is an 8-bit register, which contains the interrupt the interrupt mask (I) and four flags. The flags indicate the results of executed instructions.

Programming Model



The flags are: Half-Carry flag, Interrupt-Mask (i), Negative-Flag, Zero-Flag and Carry/Borrow-Flag.

Program Counter

- A 16-bit register that contains the address of the next instruction or operand the CPU has to fetch. The six most significant bits are not used and assumed to 0. The address the PC automatically increments to the next memory location every time an instruction or operand is fetched. Branch, jump and interrupt operations load the program counter with an address, which is not in sequential order.

Arithmetic/Logic Unit (ALU):

• The Arithemtik/Logic Unit performs the arithmetic and logical operations defined by the instruction set.

• The decoded instructions set up the ALU for the operation. Most binary arithmetic instructions are based on addition and subtraction. Multiplication is not achieved as a discrete operation. It is realized by a number of additions and shift operations within the ALU. The multiply instruction (MUL) needs 11 internal processor cycles to complete the chain of operations.

For more details especially for the Interrupt-structure, Timer and Input/Output:

see: www.motorola.com/semiconductor

MC68HC705

Instruction Set: The MCU has a set of basic instructions. The instructions can be divided into five different categories:

- Register/memory,
- Read-modify-write
- Branch,
- Bit manipulation
- Control

Addressing Modes: The CPU uses eight adressing modes for accessing data. These addressing modes define the manner in which the CPU finds the data to execute an instruction. The eight addressing modes are:

- Inherent
- Immediate
- Direct
- Extended
- Indexed no offset
- Indexed 8-Bit offset
- Indexed 8-Bit offset
- Relative

MC68HC705: Register/Memory Instructions

Most of these instructions use two operands. One operand is either the accumulator or the index register. The other operand is obtained from memory using one one of the following addressing modes:

• Immediate, Direct, Extended, Indexed (no offset), Indexed (8-bit offset), Index (16 Bit offset)

Instruction	Short Form	Adressing Modes for Operand	Cycles
Load Accumulator from Memory	LDA	6	2-5
Load Index Register from Memory	LDX	6	2-5
Store Accumulator in Memory	STA	5	4-6
Store Index Register in Memory	STX	5	4-6
Add Memory to Accumulator	ADD	6	2-5
Add Memory and Carry to Accumulator	ADC	6	2-5
Subtract memory contents from accumulator	SUB	6	2-5
Subtract memory from Accumulator with Borrow	SBC	6	2-5
AND Memory to Accumulator	AND	6	2-5
OR Memory with Accumulator	ORA	6	3-6
Exclusive OR memory with Accumulator	EOR	6	2-5
Aritmetic Compare Accumulator with memory	CMP	6	2-5
Arithmetic Compare Index Register with Memory	CPX	6	2-5
Bit Test Memory with Accumulator (Logical Compare)	BIT	6	2-5
Unsigned Multiply	MUL	-	11
Count: 15			

MC68HC705: Read-Modify-Write Instructions

These instructions read a memory location or a register, modify/test the content, and write the modified value back to the register or to the memory. The test for negative or zero (TST) instruction is an exception to the read-modify-write sequence, because ist does not write A replacement value. Used Adressing modes are:

• Inherent, Direct, Indexed (no offset), Indexed (8-bit offset), Indexed (16-bit offset)

Instruction	Short Form	Addressing Mode for Operand	Cycles
Increment	INC	5	3-5
Decrenent	DEC	5	3-5
Clear	CLR	5	3-6
Complement	COM	5	3-5
Negate (Twos Complement)	NEG	5	3-6
Rotate Left thruough Carry	ROL	5	3-5
Rotate Right through Carry	ROR	5	3-5
Logical Shift Left	LSL	5	3-5
Logical Shift Right	LSR	5	3-5
Arithemtic Shift Right	ASR	5	3-6
Arithmetic Shift Left	ASL	5	3-6
Test for Negative or Zero	TST	5	3-5
Count: 12			

MC68HC705: Branch/Jump Instructions

Instruction	Short Form	Adrressing Mode for Operand	Cycles
Branch Always	BRA	1	3
Branch Never	BRN	1	3
Branch if Higher	BHI	1	3
Banch if Bit n of M = 0	BRCLR	1	5
Branch if Bit n of M = 1	BRSET	1	5
Branch Lower or Same	BLS	1	3
Branch if Carry Bit Clear	BCC	1	3
Branch if Higher or Same	BHS	1	3
Branch if Carry Bit Set	BCS	1	3
Branch if Lower	BLO	1	3
Branch if Not Equal	BNE	1	3
Branch if Equal	BEQ	1	3
Branch if Half Carry Clear	BHCC	1	3
Branch if Half Carry Set	BHCS	1	3
Branch if Plus	BPL	1	3
Branch if Minus	BMI	1	3
Branch if Interrupt Mask is Clear	BMC	1	3
Branch if Interrupt Mask is Set	BMS	1	3
Branch if Interrupt Line is Low	BIL	1	3
Branch if Interrupt Line is High	BIH	1	3
Branch to Subroutine	BSR	1	6
Jump uncoditional	JMP	5	2-4
Jump Subroutine	JSR	6	5-7
Count 23			

These instructions branch if a particular condition is met. Otherwise, no operation is performed. Branch instructions are two byte instructions. The jump instructions have no register operand. The jump/branch instructions use the following addressing modes:

JMP/JSR:
- Direct
- Extended
- Indexed (no offset)
- Indexed (8-bit offset)
- Indexed (16-bit offset)

Branch:
Relative

By branch/jump instructions the MCU is able to interrupt the normal sequence of the program counter when a test condition is met, otherwise the branch is not performed.

MC68HC705: Control Instructions

These instructions are register reference instructions and are only used to control the operation of the processor during program execution. Control instructions use the inherent addressing mode.

Instruction	Short Form	Address Modes for Opernad	Cycles
Transfer Accumulator to Index Register	TAX	1	2
Transfe Index Register to Accumulator	TXA	1	2
Set Carry Bit	SEC	1	2
Clear Carry Bit	CLC	1	2
Set Interrupt Mask Bit	SEI	1	2
Clear Interrupt Mask Bit	CLI	1	2
Software Interrupt	SWI	1	10
Return from Subroutine	RTS	1	6
Return from Interrupt	RTI	1	9
Reset Stack Pointer	RSP	1	2
No Operation	NOP	1	2
Stop	STOP	1	2
Wait	WAIT	1	2
Count: 13			

MC68HC705: Bit Manipulation Instructions

The MCU is capable of setting or clearing any writable bit which is located in the first 256 bytes of the memory space. In this area are all port registers, port DDRs, timer, timer control, and on-chip and RAM locations. An additional feature allows the software to test and branch on the state of any bit within the 256 locations. The bit clear, bir set, and bit test and branch functions are all one-cycle instructions. These instructions use the direct addressing mode.

Instruction	Short Form	Address Modes of Operands	Cycles
Set Bit n	BSET n [0...7]	1	5
Clear Bit n	BCLR n [0...7]	1	5
Count: 2			

Addressing Modes

Immediate instructions contain a value which is to be used in an operation with the value in the accumulator or index register. Immediate instructions require no memory address and are two bytes long. The opcode is the first byte and the immediate data value is the second byte.

Example:	0200	A6 02	LDA #$12	/* Load accumulator with immediate value */

	$0200	$A6		/* MCU reads opcode $A6, load accumultaor with the value immediately follows the opcode */
	$0201	$12		/* MCU reads the immediate data $12 from location $0201 into the accumulator*/

Inherent instructions have no operand, such as return from interrupt (RTI). Some of the instructions act on data in the CPU registers, such as set carry flag (SEC) and increment accumulator (INCA). The instructions require no memory address and are one byte long (8-bit CPU).

Eample:	0200	$4C	INCA	/* Increment accumulator */

	$0200	$4C		/* MCU reads opcode $4C, increment accumulator */
				/* MCU adds one to the current accumulator value */
				/* MCU stores the new value in the accumulator, and adjusts the condition code flag bits, if necessary */

Addressing Modes

By the extended addessing mode, the effective address of the argument is contained in the two bytes following the opcode byte. Instructions with extended addressing modes are capable of referencing arguments anywhere in memory by a thre-byte instruction.

Example: 0200 C6 03 65 LDA $0365 /* Load accumulator with immediate value */

$0200	$C6	/* MCU reads opcode $C6, load accumulator with extended addressing mode */
$0201	$03	/* MCU reads $03 from location $0201. $03 is interpreted as the high-order half of an address */
$0202	$65	/* MCU reads $65 from location $0202. $65 is interpreted as the low-order half of an address */
		/* MCU calculates the complete extended address $0365 from the two values. The address is placed on the the address bus and the MCUreads the data value from location $0365 into the accumulator */

Addressing Modes

Direct addressing mode can access any of the first 256 memory addresses with only two bytes.The first byte is the opcode and the second byte is the low byte of the oparand address. In the direct mode, the CPU uses automatically $00 as the high byte of the operand address. BRESET and BRCLR are three-byte instructions that use direct addressing to access the operand and relative addressing to specify a branch destination.

Example:	0200	B6 E0	LDA $E0	/* Load accumulator from a direct page address */

	$0200	$B6		/* MCU reads opcode $B6, load accumulator with direct addressing mode */
	$0201	$E0		/* MCU reads $E0 from location $0201. This $E0 is interpreted as the low order half of an address in the direct page ($0000 to $00FF)*/
				/* MCU builds the complexe direct address $00E0 from the assumed high-order value $00 and thepreviously read low-order address value. This address is set on the bus and the MCU reads the data value from location $00E0 into the accumulator */

Addressing Modes

In the indexed mode with no offset the effective address of the argument is contained in the 8-bit index register. This addressing mode can access the first 256 memory locations. The instructions are only one byte long. This mode is only used to move a pointer through a table or to hold the address of a frequently referenced RAM or I/O location

Example: 0200 F6 LDA 0,X /* Load accumulator from address pointed to by X */

$0200 $F6 /* MCU reads opcode $F6, load accumulator with indexed addressing mode with no offset */

/* MCU builds a complete address by adding $0000 to the contents of the 8-bit index register X */

/* The builded address is set on the bus and the MCU reads the data value from that location into the accumulator */

Addressing Modes

In the indexed mode with 8-bit offset, the effective address is the sum of the contents of the unsigned index register X and the unsigned byte, which followes the opcode. This mode is usefull for selecting the Ith element in an n element table. With the two-byte instruction , I is typically in X with the address of the beginning of the table in the instruction.

Example:	0200	E6 05	LDA 5,X	/* Load accumulator with the 6th item in table starting at X */

	$0200	$E6		/* MCU reads opcode $F6, load accumulator with indexed 8-bit offset addressing mode */
	$0201	$05		/* MCU reads 8-bit offset ($05) from address $0201 */
				/* MCU builds a complete address by adding the value just read ($05) to the contents of the 8-bit index register X */
				/* This address is placed on the address bus and the CPU reads the data value from that location to the accumulator */

Addressing Modes

In the indexed mode with 16-bit offset, the effective address is the sum of the contents of the unsigned index register X and the two unsigned byte, which followes the opcode. This mode is usefull for selecting the Ith element in an n element table. With the three-byte instruction , I is typically in X with the address of the beginning of the table in the instruction.

Example: 0200 D6 03 77 LDA $377,X /* Load accumulator with the Xth item in table at $0377 */

$0200	$D6	/* MCU reads opcode $D6, load accumulator using indexed 16-bit offset addressing mode */
$0201	$03	/* MCU reads high-order half of 16-bit base address ($03) from address $0201*/
$0202	$77	/* MCU reads low-order half of 16-bit base address ($77) from address $0202*/
		/* MCU builds a complete address by adding the contents of the 8-bit index register (X) to the 16-bit base address previously read */
		/* The address is set on the bus and the MCU reads the data value from that location into the accumulator */

Addressing Modes

The relative addressing mode is exclusively used in branch instructions. In relative addressing mode, the content of the 8-bit signed byte following the opcode (offset). Is added to the PC if, and only if, the branch conditions are true.

Example:	0200	27 rr	BEQ DEST	/* Branch to DEST if Z=1 (equal or zero) */

$0200	$27	/* MCU reads opcode $27, branch if Z=1. The Z condition code will be 1 if the result of the previous arithmetic or logical operation was zero *7
$0201	$rr	/* MCU reads the offset value $rr from loaction $201. After this cycle the program counter (PC) is pointing at the first byte of the next instruction ($0202(*/ ($03) from address $0201*/
		/* If Z-bit is zero, no action takes place in this cycle, and the program will just continue to the next instruction at $0202. If Z-bit is one, the MCU will add the signed offset $rr to the present value in the PC to get the address of the branch destination. This causes program execution to start from the addres DEST */

MC68HC705:

Max count of different instructions is 65; with the various addressing modes the MCU works with 207 various instructions.

Why ??

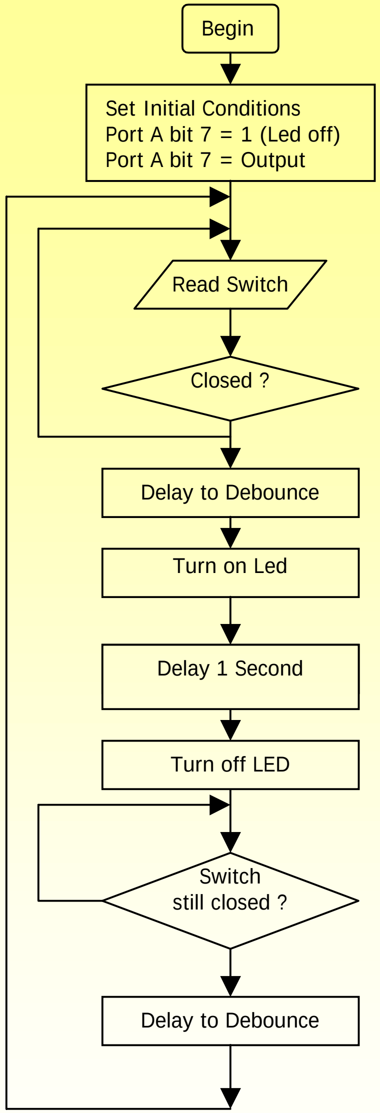
Source: Understanding Small Microcontrollers; James M. Sibigtroth, Motorola CSIC Microcontroller Division,

Simple Example for MC68HC05

- Read state of switch at port A bit-0;   1 = closed
- When switch closed, light LED on for about 1 second, LED on
- Turn LED off (port A bit-7 = 0)
- When port A bit-7 = 0; Wait for switch releases, then repeat.
- Debounce switch for 50 ms on & off

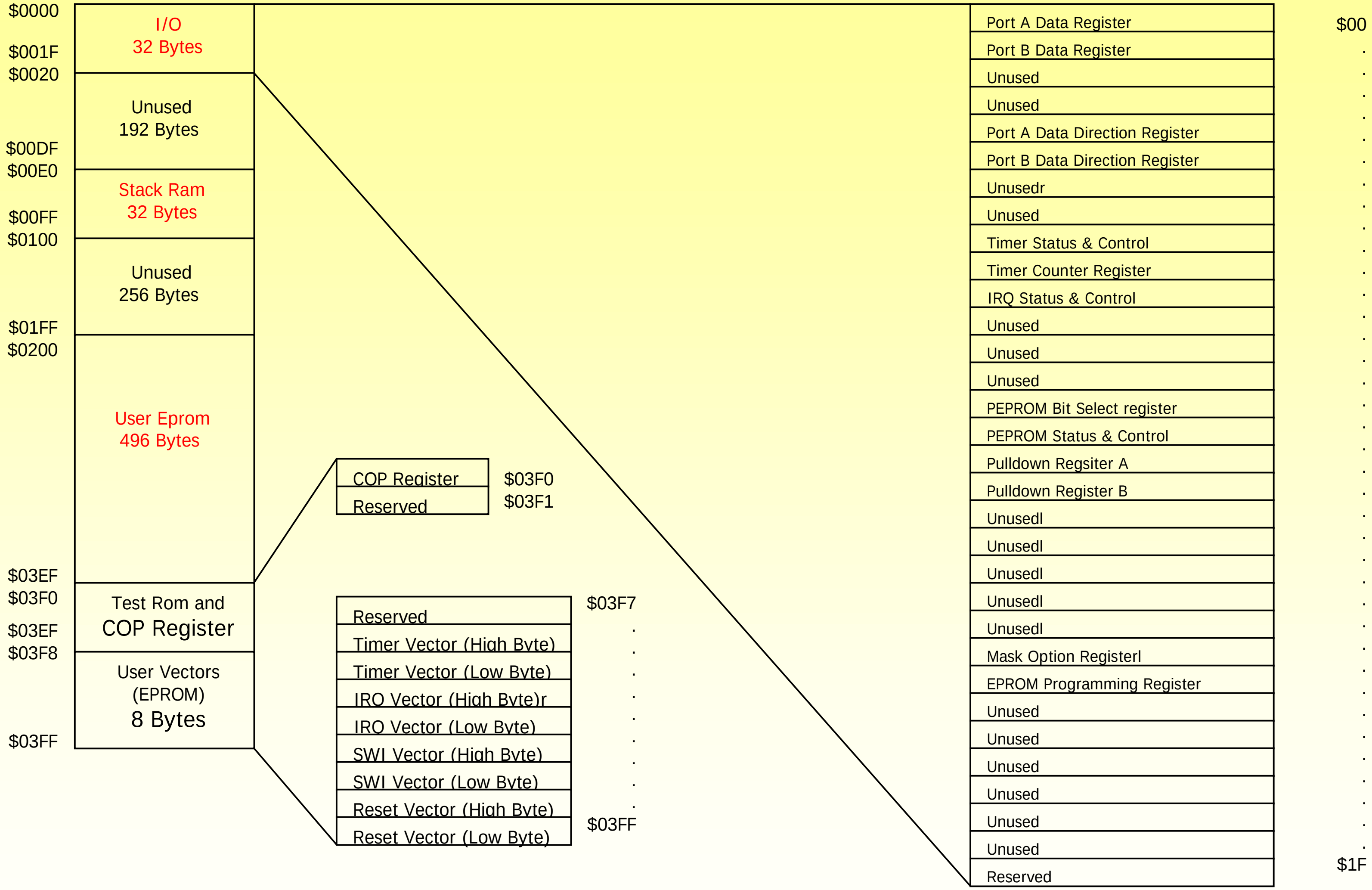
•Note: The timing is based on the instruction execution times.

Example:




```
$BASE       10T

PORTA       EQU   $00             /* Direct address port A*/
DDRA        EQU   $04             /* Data direction control, port A*/
TEMP1       EQU   $E4             /* One byte scratch storage loaction */

            ORG $0200             /* start at this location */

INIT        LDA  #$80             /* Begin initialisation */
            STA   PORTA           /* LED is off */
            STA   DDRA            /* Set port A bit-7 as output */

TOP         LDA   PORTA           /* Read switch at LSB of port A */
            AND  #$01             /* Test bit-0 */
            BEQ  TOP              /* Loop till bit-0 = 1*/
            JSR   DLY50           /* Delay about 50 ms to debounce */
            BCLR  7, PORTA        /* Turn on LED (bit-7 = 0 */
            LDA   #20             /* Decimal 20 assembles to $14 */
DLYLP       JSR   DLY50           /* Delay 50 ms */
            DECA                  /* Loop counter fro 20 loops */
            BNE   DLYLP           /* 20 times (20-19,...1-,0) */
            BSET  7, PORTA        /* Turn LED off */
OFFLP       BRESET 0, PORTA, OFFLP  /* Loop here till switch off */
            JSR   DLY50           /* Debounce Release */
            BRA   TOP             /* Look and wait for next swicth closed */

*/  Subroutine */

DLY50       STA   TEMP1           /* Save accunulator in RAM*/
            LDA   #65             /* Do outer loop 32 times */
OUTLP       CLRX                  /* X used as inner loop count */
INNRLP         DECX               /* (0-FF, FF-FE, ....1-0), 256 times ! */
               BNE    INNRLP      /* 6 cycles * 256 * 500 ns/cycle = 0.768 ms*/
            DECA                  /* (65-64, ....., 1-0) */
            BNE   OUTLP           /* 154 cycles * 65*500ns/cycle = 50.212 ms*/
            LDA   TEMP1           /* Recover saved Accumulator val*/
            RTS                   /* Return */
```

A memory map is a pictorial representation of the total MCU memory space.



Typical Memory Map

Sensors	Data Acquisition	Memory	Processor	Output	Actuators
Voltage, Current	A/D	RAM	Micro-processor	D/A	Loud-speaker
Temp-erature	Codec	Dual Port RAM	Micro-controller	Digital	Relay, Led Lamp
Light, Pressure, Noise	Digital (Fifo, Latch Register)	Multi Port RAM	DSP	Display	Switch
Audio, Ultrasound		Shared RAM	Transputer	SSI/SPI	Motor
Position, Velocity Acceleration		Fifo	Core-Processor	CAN, Profibus	....
Human Interfaces			DSP Multi-Proc essor System	Network (Ethernet)	
....				Bus-Interfaces	

A more generall view to different functional blocks used in embedded systems.

31.10.02

Part of next lectures

What are specific DSP applications ?

- Wireless handsets and personal communication systems
- Portable audio players
- Personal medical devices
- Digital cameras
- Internet/information purposes
- Power-efficient multichannel telephony systems
- Speech recognition and decoding
- Line or Acoustic Echo cancellation; noise cancellation and reduction
- Modulation and demodulation
- Image and audio compression and decompression
- speech encryption, decryption
- speech recognition, speech analysis
- industrial, automotive, consumer white goods and office market
- fast instrumentation and measurement devices

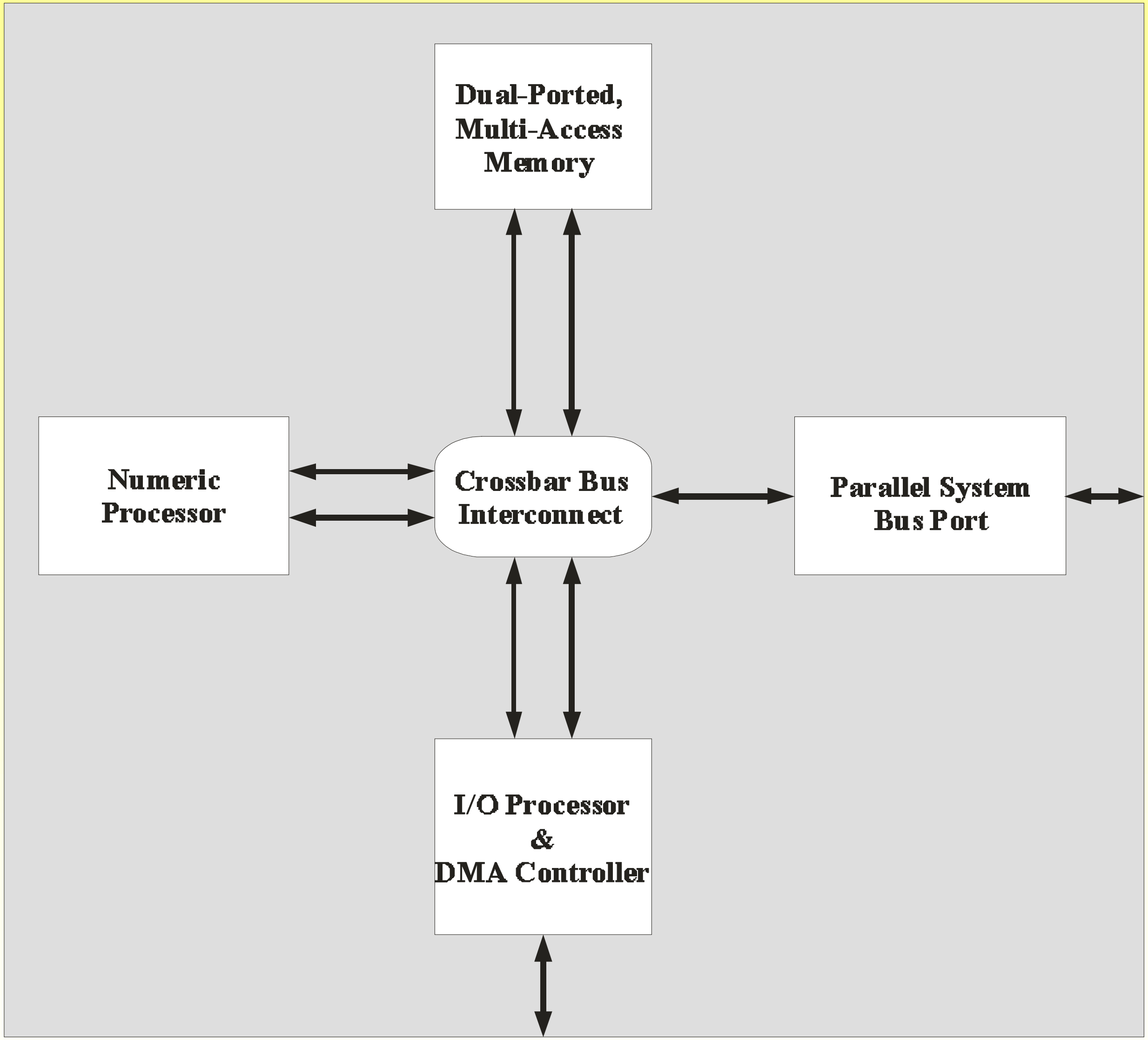
Characteristics: of DSPs

- Single Chip Devices
- Low power by low core voltage (1.8 Volt !)
- Multiply-accumulate units
- Multiple access memory architecture
- Specialized addressing modes: auto-modify addressing, circular addressing, bit-reverse addressing
- Fast interrupts
- Predicated execution in one cycle
- Hardware loops / zero-overhead loops
- Fast link busses for multiprocessing
- Restricted interconnectivity between registers and functional units

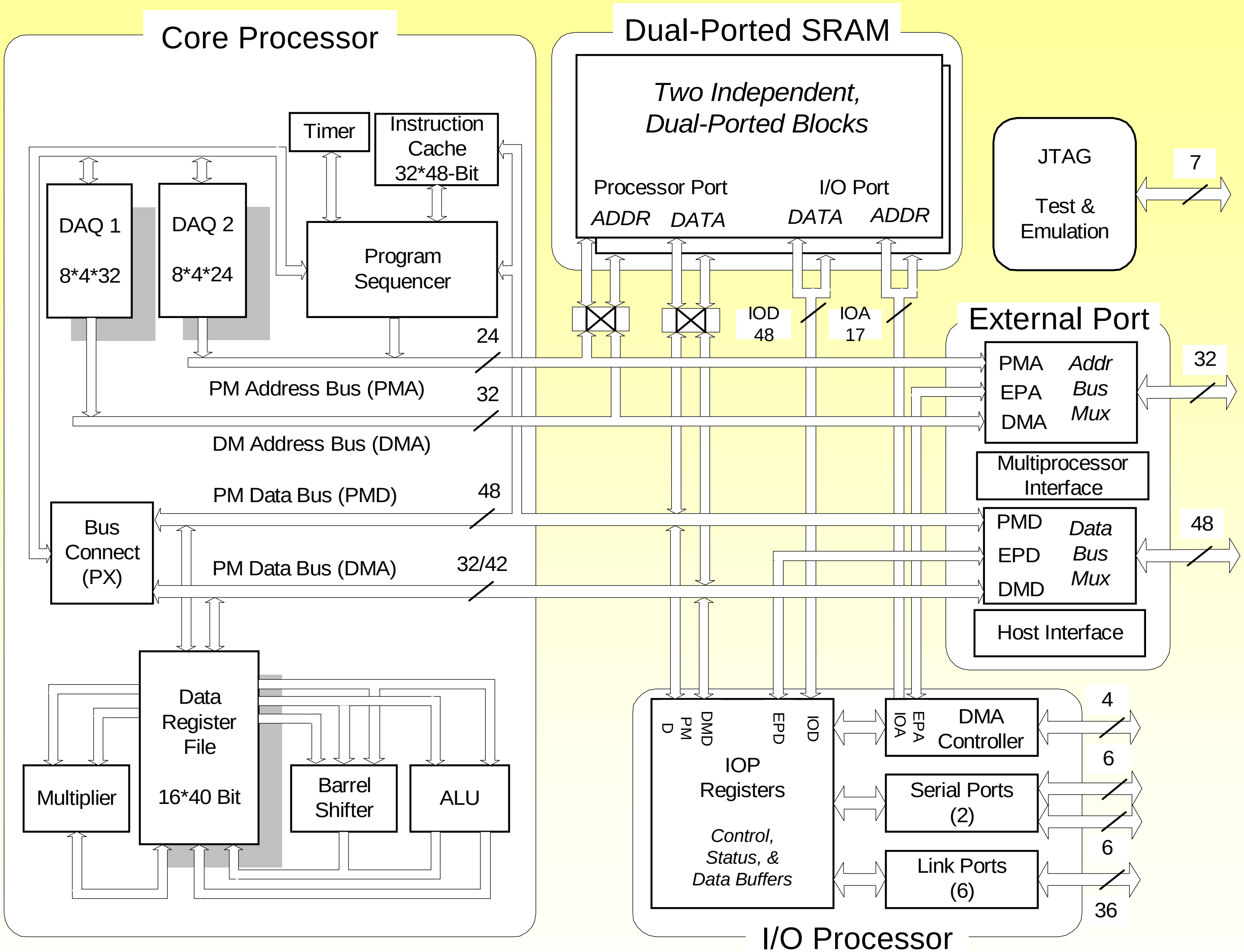
DSP SHARC

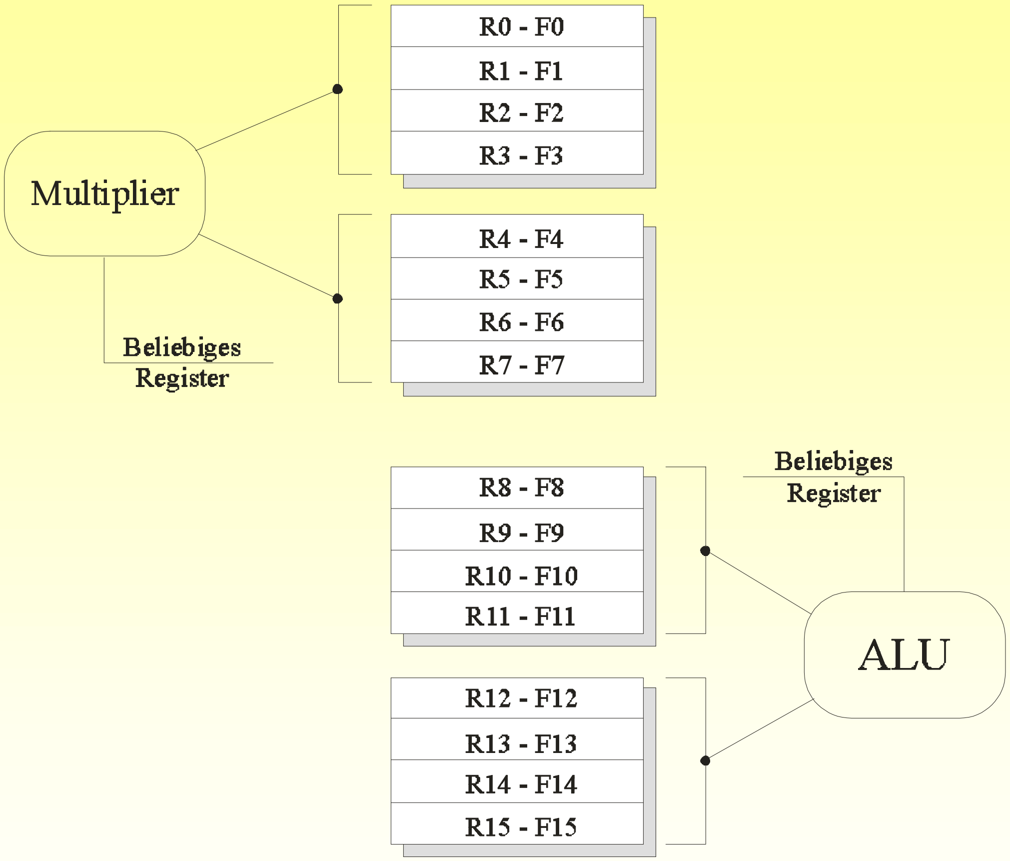
The ADSP-2106x SHARC (**S**uper **H**arvard **Ar**chitecture **C**omputer) has extensive numerical power for signal processing applications. All instructions are executed in one cycle. With the dual-ported on chip SRAM and the integrated I/O peripherals, as SSI interface, 4 Bit-Link interface, direct-memory-access and fas interrupt logic, it is a powerfull single chip processor. Four independant busses and a high speed crossbar switch are integrated.

- 120 Mflops / 80 Mips
- 2 MBit Dual-Port-RAM (intern)
- 6 parallel Link-Ports (40 MByte/s)
- 2 Serielle Schnittstellen (40 MBit/s)
- Host-Interface
- Interface for multiprocessing of up to 6 DSPs
- External Hardware-Interrupts
- I/O Signals


Dual-Ported,
Multi-Access
Memory
Numeric
Processor
Crossbar Bus
Interconnect
Parallel System
Bus Port
I/O Processor
&
DMA Controller

SHARC Core Processor





Register Rx: 32 Bit Integer
Register Fx: 32 Bit Floating Point
Shadow register for fast context switching

Example:

F3 = F3 * F2, F8 = F10 + F12;

Instruction Set Reference

- Compute and Move or Modify Instructions, which specify a compute operation in parallel with one or two data moves or an index register
- Program Flow Control instructions, which specify vaious types of branches, calls returns and loops. Some of these instructions may also specify a compute operation and/or a data move
- Immediate Data Move instructions, which use immediate instruction fields as operands, or use immediate instruction fields for addressing.
- Miscellaneous instructions, such as bit modify and test, no operation and idle

Instruction Set Notation

;	Semicolon terminates the instruction
,	Comma separates parallel operations in an instruction
|option1|	List of options in vertical bars; one is a possible choice
compute	ALU, MUL, SHIFTER or multifuntion opoeration
shiftimm	Shifter immediate operatio
condition	Status conditio
termination	Termination condition of a loop
ureg	universal register
sreg	system register
dreg	Data registerin register file (R0-R15, F0-F15)
Ia	DAG1 index register (I0-I7)
Mb	DAG1 modify register (M0-M7)
Ic	DAG2 index register (I8-I15)
Md	DAG2 modify register (M8-M15)
<datak>	k-bit immediate data value
<addrk>	k-bit immediate address value
<reladdrk>	k-bit immediate PC-relative address value
(DB)	Delayed branch
(LA)	Loop abort (pop loop and PC stacks on branch)
(CI)	Clear interrupt

Compute & Move or Modify Instructions

<table>
<tr><th>Nr.</th><th colspan="7">Instruction</th><th>Addressing mode</th></tr>
<tr><td>1</td><td></td><td>compute</td><td>,</td><td>|DM(Ia,Mb)=dreg1|
|dreg1=DM(Ia,Mb)|</td><td>,</td><td>|PM(Ic,Md)=dreg2|
|dreg2=PM(Ic,Md)|</td><td>;</td><td>Indirect addressing ‚post-modify‘</td></tr>
<tr><td>2</td><td>If condition</td><td>compute</td><td>;</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3a</td><td>If condition</td><td>compute</td><td>,</td><td>|DM(Ia,Mb)=ureg|
|PM(Ic,Md)=ureg|</td><td>;</td><td></td><td></td><td>Indirect addressing ‚post-modify‘</td></tr>
<tr><td>3b</td><td>If condition</td><td>compute</td><td>,</td><td>|DM(Ma,Ib)=ureg|
|PM(Mc,Id)=ureg|</td><td>;</td><td></td><td></td><td>Indirect addressing ‚pre-modify‘</td></tr>
<tr><td>3c</td><td>If condition</td><td>compute</td><td>,</td><td>|ureg=DM(Ia,Mb)|
|ureg=PM(Ic,Md)|</td><td>;</td><td></td><td></td><td>Indirect addressing ‚post-modify‘</td></tr>
<tr><td>3d</td><td>If condition</td><td>compute</td><td>,</td><td>|ureg=DM(Ma,Ib)|
|ureg=PM(Mc,Id)|</td><td>;</td><td></td><td></td><td>Indirect addressing ‚pre-modify‘</td></tr>
<tr><td>4a</td><td>If condition</td><td>compute</td><td>,</td><td>|DM(<data6>)=dreg|
|PM(<data6>)=dreg|</td><td>;</td><td></td><td></td><td>Immediate addressing mode</td></tr>
<tr><td>4b</td><td>If condition</td><td>compute</td><td>,</td><td>|DM(<data6>,Ia)=dreg|
|PM(<data6>,Ic)=dreg|</td><td>;</td><td></td><td></td><td>Immediate addressing mode ‚pre-modify‘</td></tr>
<tr><td>4c</td><td>If condition</td><td>compute</td><td>,</td><td>|dreg=DM(<Ia,<data6>)|
|dreg=PM(Ic,<data6>)|</td><td>;</td><td></td><td></td><td>Immediate addressing mode ‚pre-modify‘</td></tr>
<tr><td>4d</td><td>If condition</td><td>compute</td><td>,</td><td>|dreg=DM(<data6>,Ia)|
|dreg=PM(<data6>,Ic)|</td><td>;</td><td></td><td></td><td>Addressing mode: ‚Immediate-pre-modifier‘</td></tr>
<tr><td>5</td><td>If condition</td><td>compute</td><td>,</td><td>ureg1=ureg2</td><td>;</td><td></td><td></td><td>Register transfer</td></tr>
<tr><td>6a</td><td>If condition</td><td>shiftimm</td><td>,</td><td>|DM(Ia,Mb)=dreg|
|PM(Ic,Md)=dreg|</td><td>;</td><td></td><td></td><td>Indirect addressing ‚post-modify‘</td></tr>
<tr><td>6b</td><td>If condition</td><td>shiftimm</td><td>,</td><td>|dreg=DM(Ia,Mb)|
|dreg=PM(Ic,Md)|</td><td>;</td><td></td><td></td><td>Indirect addressing ‚post-modify‘</td></tr>
<tr><td>7</td><td>If condition</td><td>shiftimm</td><td>,</td><td>|Modify(Ia,Mb)|
|Modify(Ic,Md)|</td><td>;</td><td></td><td></td><td>Modify address (Ia=Ia+Mb)
(Ic=Ic+Md)</td></tr>
</table>

Program Flow Control Instructions

<table>
<tr><th>Nr.</th><th></th><th></th><th></th><th></th><th></th><th>Addressing Mode</th></tr>
<tr><td>8a</td><td>If condition</td><td>JUMP</td><td>|<addr24>|
|(PC,<reladdr24>)|</td><td>|(DB)|
|(LA)|
|(CI)|
(|DB,LA)|
|(DB,CI)|</td><td>;</td><td>Direct addressing
Relative addressing</td></tr>
<tr><td>8b</td><td>If condition</td><td>CALL</td><td>|<addr24>|
|(PC,<reladdr24>)|</td><td>|(DB)|</td><td>;</td><td>Direct addressing
Relative addressing</td></tr>
<tr><td>9a</td><td>If condition</td><td>JUMP</td><td>(Md,Ic)
(PC,<reladdr6>)</td><td>|(DB)|
|(LA)|
|(CI)|
(|DB,LA)|
|(DB,CI)|</td><td>, |compute| ;
|ELSE compute|</td><td>Indirect ‚pre-modify‘
Relative addressing</td></tr>
<tr><td>9b</td><td>If condition</td><td>CALL</td><td>(Md,Ic)
(PC,<reladdr6>)</td><td>|(DB)|</td><td>, |compute|;
|ELSE compute|;</td><td>Indirect ‚pre-modify‘
Relative addressing</td></tr>
<tr><td>10</td><td>If condition</td><td>JUMP</td><td>(Md,Ic), ,
(PC,<reladdr6>),</td><td>ELSE</td><td>|compute, DM(Ia,Mb)=dreg| ;
|compute, dreg=DM(Ia,Mb)|</td><td>Indirect ‚pre-modify‘
Relative addressing</td></tr>
<tr><td>11a</td><td>If condition</td><td>RTS</td><td>|(DB)|, ,
|(LR)|,
|(DB,LR)|,</td><td>|compute| ;
|ELSE compute|</td><td></td><td>Direct addressing
Relative addressing</td></tr>
<tr><td>11b</td><td>If condition</td><td>RTI</td><td>(DB), ,</td><td>|compute|; ;
|ELSE compute|</td><td></td><td>Direct addressing</td></tr>
<tr><td>12</td><td>LCNTRL=</td><td>|<data16>| ,
|ureg|</td><td>DO |<addr24|
|(PC,<reladdr24>)|</td><td>UNTIL LCE ;</td><td></td><td>Direct addressing
Relative addressing</td></tr>
<tr><td>13</td><td>DO</td><td>|<addr24>|
|(PC,<reladdr24>)|</td><td>UNTIL termination ;</td><td></td><td></td><td>Direct addressing
Relative addressing</td></tr>
</table>

Immediate Move Instructions

Nr.		Addressing Mode
14a	|DM(<addr32>)| = ureg ; |PM(<addr24>)|	Direct addressing
14b	ureg=|DM(<addr32>)| ; |PM(<addr24>)|	Direct addressing
15a	|DM(<data32>)| = ureg ; |PM(<data24>)|	Direct addressing
15b	ureg=|DM(<data32>)| ; |PM(<data24>)|	Direct addressing
16	|DM(Ia,Mb)| = <data32> ; |PM(Ic,Md)|	Direct addressing
17	ureg = <data32> ;	Direct addressing

Miscellaneous Instructions

Nr.	Instruction
18	BIT |SET| sreg <data32> ; |CLEAR| |TGL| |TST| |XOR|
19a	MODIFY |(Ia,<data32>| ; |(Ic,<data24>|
19b	BITREV |(Ia,<data32>| ; ; |(Ic,<data24>|
20	|PUSH| LOOP , |PUSH| STS , |PUSH| PCSTK , FLUSH ; |POP| |POP| |POP| CACHE
21	NOP ;
22	IDLE ;
23	CJUMP |function| (DB) ; |(PC,<reladdr24>|
24	RFRAME ;

Data Addressing

Overview

•2 data address generators (DAG1 and DAG2)
• indirect address access, address indirect by the content of a DAG
• DAG1 addresses 32-bit on data bus DM
• DAG2 addresses 24-bit on program bus PM
• Additional alternate (secondary) register for fast context switching

• Special support for signal processing applications
  • Circular data buffers
  • Bit-reversing

• Each has 4 types of special registers:

• Index (I)	= pointer to memory	[DAG1: I0-I7] [DAG2: I8-I15]
• Modify (M)	= increment/decrement value	[M0-M7] [M8-M15]
• Base (B)	= base address of a circular buffer	[B0-B7] [B8-B15]
• Length (L)	= length of a circular buffer	[L0-L7] [L8-L15]

DAG operation

• address output (pre-modufy or post-modify)
• modulo addressing (circular buffers)
• addressing in bit-reverse order

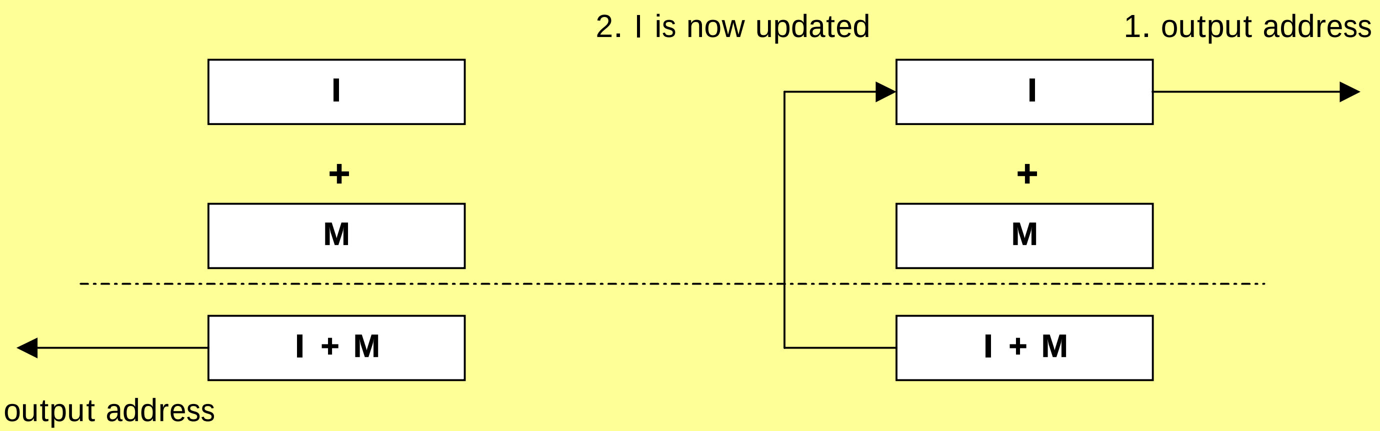
Pre-Modify versus Post-Modify

Pre-Modify: No update of I-register

PM(Mk,Ik)
DM(Mk,Ik)

Post-Modify: Update of I-register

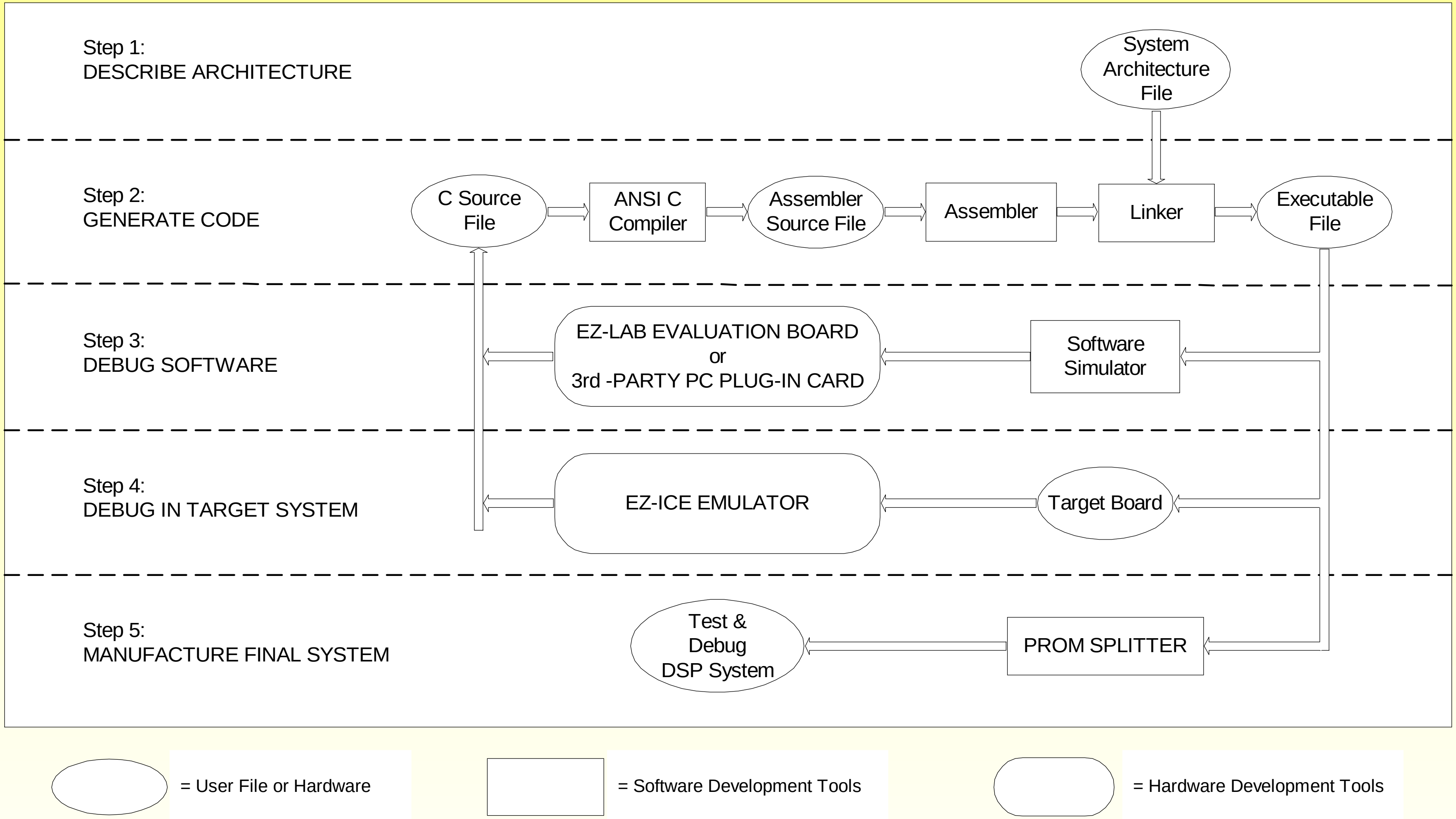
PM(Ik,Mk)
DM(Ik,Mk)



Modifier Instructions

- **R6 = PM(I12,M11);** Indirect addressing PM-memory with post-modify
  - R6 = content of I12
  - I12 = I12 + M11

- **R6 = PM(M11,I12);** Indirect addressing PM-memory with pre-modify
  - R6 = content of (I12 + M11)
  - I12 is not changed
- **DM(M1,I2) = TCOUNT**; Indirect addressing DM-memory
  - Store TCOUNT in DM address (I2+M1)
- **R2 = DM(0x40000012, I1);** Immediate 32-bit modify: address = I1 + 0x40000012
- **R6 = F1 + F3, PM(I8,0x0A)=ASTAT;** Immediate 6-bit modify: address=I8, I8=I8+0x0A



Development Environment