2 Major Architectures of Microcomputers
- Princeton (Von Neumann) – Single Mem. (program code & data)
- Harvard – 2 seperate memories (1) data (2) Program code ONLY (Instructions)
  - (Supercomputer)
  - (Everyday life)

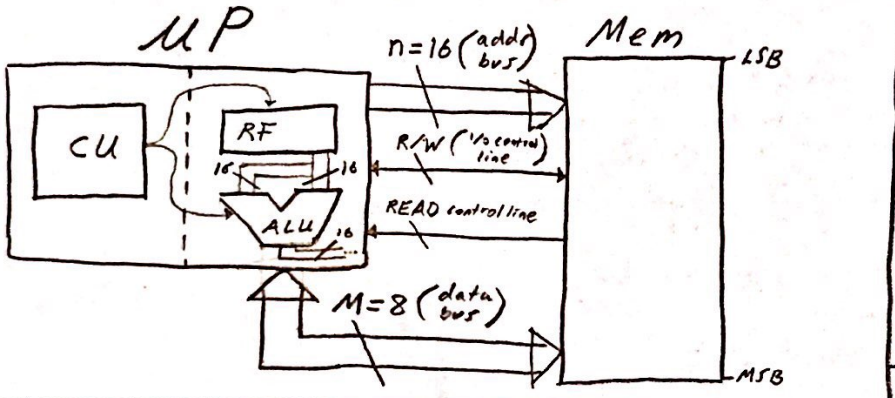
Microcomputer (Computer) – MP, control unit (CU), registers, memory, I/O devices
- MP: data processor
- control unit: brain (gives orders)
- registers: (on chip memory)
- memory: RAM (volatile), ROM (non-volatile)

MP – Single chip dedicated to data processing
- processing unit (ALU)
- control unit (CU)
- registers (on chip memory)

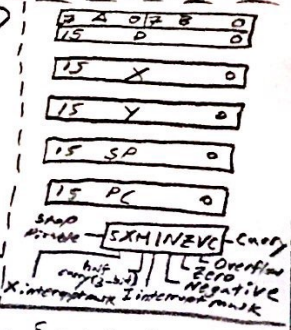
MC – Single chip computer (everything integrated into single chip)

$ – hex
% – binary
– decimal
(nothing)

HCS12 has 16-bit CPU & 64K Addr. Space



*HCS12 has 6 registers inside of RF: (registers use D FA to store memory values)
(1) A & B OR D
(2) X (index reg)
(3) Y (index reg)
(4) SP (stack ptr)
(5) PC (program ctr)
(6) CCR (condition code reg)



*(2) is 8-bit accumulator A&B OR 16-bit accumulator D

Memory Size: $2^n \times M = 2^{16} \times 8$ (64K x 1-Byte ⇒ 64KB)
(write answer like this!!!)
⇔ address space * data bus

16-bit Addr Bus
8-bit Data Bus
{Memory Operations} {read (1) write (0)}

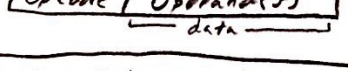
Bit-Groups:
- Nibble – 4-bits
- Byte – 8-bits
- Word – 16-bits
- DoubleWord – 32-bits

Addr. Space: 2^n = max # of addresses (2 is for binary) uniquely

Data Bus: M = Memory Word Length (memory bit)

RISC {reduced instruction set computer} fixed length (MIPS, ARM)
CISC {complex instruction set computer} variable length (HCS12, x86)

Binary Instruction:



operands
A + B
↑
Op Code

Assembler turns higher level language code into binaries (data machine code)

Instruction Cycle: (HCS12 instruction length varies between 1-6 byte) ←

Instruction will have 1-2 bytes of opcode & 0-5 byte of operand addr. info
⇒ if 1st byte of Opcode is $18, it is 2-byte opcode

PHASES/stages or segmentation

(1) Fetch Phase	fetch instruction (opcode & operand(s))
(2) Decode Phase	decode opcode of instruction
(3) Fetch Operand(s) Phase	fetch operand(s) from data memory (to CPU registers)
(4) Execute Phase	perform operation given by opcode
(5) Store/Writeback Phase	store results in registers or data memory

*MACROS:
like subroutine but uses more memory, reusing (copying) code instance for each future occurence

Scanned with CamScanner

0101	0110 ← machine code
Operation	Operand
Opcode	Data
ADD	A,B ← Assembly code

(a.k.a MNEMONIC code)

ASM Instruction Format:

(commonly used for BRANCH/JUMP) LABEL (logical ptr to instruction addr ⇒ memory location) | (ADDA, STAA, LDDA) MNEUMONIC (logical representation of machine code) | OPERAND1, OPERAND2 (data to be operated on, can be CPU register or memory)

Addressing Mode: Tells processor how to access operands

Mnemonic	Operation
LDAA	Load Accum A
LDAB	" " B
INCA	inc Accum A
ABA	add accumulators
STAA	store A in Mem
STAB	store B in Mem
WAI	wait for interrupt

high byte → 01 48₁₆ ← low byte

these Opcodes assume memory loc. is low addr ($000-$00FF) so high byte is always $00

Simple Program:

		# of bytes
LDAA	#10	2
LDAB	#L5	2
ABA		1
STAA	$C411	3
WAI		1

total size of program: 9 Bytes

PC contains next instruction's address

Assembler Directives: generate NO machine code!! (takes up zero bytes in program size)

EQU, ORG (PC location defined), RMB, FCB, FCC, FDB

#INCLUDE, END

default: $B600

* ORG defines program (PC) location in memory (1st byte of machine code loc.)

DS (define storage) size: B, W → label before DS[.<size>] <n>

DC (define constant) B,W,L → " " <expression>

DCB (" " block) B,W,L → " " <count><value> (byte, word, longword)

Syntax: <LABEL> FCC <"String"> for FCC (form const. characters)

Transfer A to B	TAB	INH addr modes
Transfer B to A	TBA	
Clear Accum A	CLRA	
Clear Accum B	CLRB	

Addressing Modes for HCS12 (7)

* Effective Addr (EA): address where OPERAND is stored

	[Instruction Length]	[Effective Addr EA]	[Instruction]	[Notes]	[Example]
(1) Inherent (INH)	1-byte (ONLY!)	N/A	OPCODE ONLY	INCA - incr A; INX - incr X; PSH - push; PUL - pull	ADDA #
(2) Immediate (IMM)	2-3 bytes	EA = starting addr of instr. +1; EA=PC+1	#HASHTAG! giving immediate values		ADDA #40
(3) Direct (DIR)	2 bytes	EA = 1 (2nd Byte)	OPERAND ADDRESS (NOT IN MEMORY!) (uses last byte to specify 8-bit operand addr of instruction)	Absolute Addresses	ADDA $40
(4) Extended (EXT)	3 bytes	EA = (2nd & 3rd Byte)	MEMORY SPACE ADDRESS (uses last 2 bytes to specify 16-bit addr for MEMORY!)	Absolute Addresses	ADDA $0040
(5) Relative (REL)	2-3 bytes	EA = PC + (2nd & 3rd) bytes of instruction	BRANCH ONLY (not JMP) ⇒ BEQ offset (target addr = PC+offset)	BRA - branch always; BEQ - branch =; BNQ - branch !=	
(6) Indexed (IND)		EA = offset + contents of index register	Contents of Index reg. act as address (index reg.: X, Y, SP, PC) (base register)	* Can use A,B as the 8-bit offset	ADDA X,1
(7) Index Indirect			[ ] SQUARE BRACKETS! (double pointer)	16-bit offset or accumulator D as offset	ADDA [X,1]

Indexed Addressing Mode (6&7) can use pre/post increment/decrement (±#, #±)

CLR - all bits to Ø
SET - all bits to 1
} Byte-level (set all bits to #)

DEC decrement { max length 3 bytes; min length: 1 byte (for instruction) }

(6) IND vs (7) IND Indirect

X = $1000

(6) IND	(7) IND Indirect
LDAA $40,X (offset, base register)	LDAA [$40,X]
⇒ A ← M[X+$40]	⇒ A ← M[M[X+$40]]
A ← M[$1040]	A ← M[M[$1040]]
	A ← M[$2010]
A = $20	A = $A0

Address	Contents
$1040	$20
$1041	$10
$2010	$A0

Whole address (16-bit)

BCLR
BSET } Bit-level ⇒ use mask to indicate bits to invert

Ex:
BCLR $4043, %1100 1100 = 0010 0000
BSET $4043, %0101 1101 = 1111 1101

*Note: M[$4043] = 1010 1100

Scanned with CamScanner

Signed (+&-): V-flag

8-bit range: $[-2^7, 2^7+1] \Leftrightarrow [-128, 127]$

16-bit range: $[-2^{15}, 2^{15}+1] \Leftrightarrow [-32768, 32767]$

$V = C_7 \oplus C_6 \Rightarrow$ if $=\emptyset$, VALID result

(C_7: Carry out of MSB; C_6: Carry into MSB)

[if V-flag = 1, signed addition is incorrect]

Unsigned (+): C-flag

8-bit range: $[\emptyset, 2^8-1] \Leftrightarrow [0, 255]$

16-bit range: $[\emptyset, 2^{16}-1] \Leftrightarrow [0, 65536]$

$\emptyset$, VALID result

1, INVALID result (unsigned overflow)

Addition: ADDA $2,Y+ ⇔ ADDA n,X+ ⇒ A ← A+M[X], X' = X+n

$A = \$9A$, $Y = \$4000$

$A \leftarrow A + M[Y]$

$A \leftarrow \$9A + M[\$4000]$

$A \leftarrow \$9A + \$20$

$Y' = Y + \$2$

$Y' = \$4000 + \$2$

$Y' = \$4002$

M

$ 4000	$ 20
$ 4001	$ 30
$ 4002	$ 40
$ 4003	$ 55

FLAGS	
C	0
V	0
N	1
Z	0

$\emptyset (=C_6)$

1001 1010 ($9A)

\+ 0010 0000 ($20)

$\emptyset$ | 1011 1010 ($BA)

C_7

C-flag = $\emptyset$ ⇒ VALID ✓ (if unsigned operation)

V-flag = $C_7 \oplus C_6 = \emptyset$ ⇒ VALID ✓ (if signed operation)

2's Complement: for a signed

$BA = 1011\,1010$ ⇒ negative #

invert: 0100 0101

add+1: + 1

$-(0100\,0110)$

$+64 + 6 = 70$ ⇒ −70

Subtraction: SUBA $2,Y+ ⇔ SUBA n,X+ ⇒ A ← A−M[X], X' = X−n

$A = \$BA$, $Y = \$4000$

$A - M[Y] \rightarrow A + (M[Y])' + 1 \rightarrow A + (\$20)' + 1$

$\Rightarrow A \leftarrow \$BA + (\$20)' + \$01$

$(C_6)'$ 1

1011 1010 ($BA)

1110 0000 ($\overline{\$20}$ + 1)

0 | 1001 1010

$C_7 (= C')$

* Use 2's C addition:
  $A - B \rightarrow A + B' + 1$
  (w/ B' = compliment of B)

* Invert carry out bits to indicate a borrow

☆ for subtraction C_7 is the complement of C (C') in the carry slot, indicating borrow (eg., $C = \emptyset \rightarrow C' = b = 1 = C_7$)

FLAGS	
C	0
V	1
N	1
Z	0

⇒ V-flag = $C_7 \oplus C_6 = 0 \oplus 1 = 1$ ⇒ INVALID

(pos#) + (neg#) ⇒ ALWAYS VALID

(no signed overflow, V-flag = $\emptyset$)

(pos#) − (pos#) OR (neg#) − (neg#) ⇒ ALWAYS VALID

(no signed overflow, V-flag = $\emptyset$)

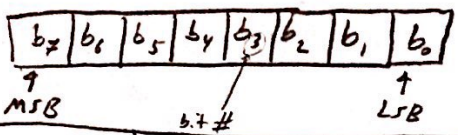
(pos#) + (pos#) OR (neg#) + (neg#) ⇒ POSSIBLE SIGNED OVERFLOW

(pos#) − (neg#) ⇒ POSSIBLE SIGNED OVERFLOW

Scanned with CamScanner

Binary	(base-2)	101010_2
Decimal	(base-10)	245_{10}
Hexadecimal	(base-16)	$A5_{16}$

8-bit representation



Bin 2 Dec:

if bit3, $b_3 = 1 \rightarrow$ dec val $= 1 * 2^3 = 8$

Dec Val $= N = \sum_{i=0}^{n} b_i \times 2^i = \ldots$

Dec2Bin: $93_{10} \Leftrightarrow 01011101_2$

	Remainder
$93/2 \rightarrow$	1 (LSB)
$46/2 \rightarrow$	0
$23/2 \rightarrow$	1
$11/2 \rightarrow$	1
$5/2 \rightarrow$	1
$2/2 \rightarrow$	0
$1/2 \rightarrow$	1

0 → (if needed, this is MSB)

Dec2Hex: $34490_{10} \Leftrightarrow 86BA_{16}$

	R
$34490/16 \rightarrow$	10 (LSB)
$2155/16 \rightarrow$	11
$134/16 \rightarrow$	6
$8/16 \rightarrow$	8 (MSB)

0

Hex2Bin: (4-bit bin ⇔ 2-bit Hex)

$0 = 0000_2$ $\quad A = 1010_2$

$9 = 1001_2$ $\quad F = 1111_2$

Hex2Dec: $86BA_{16} \Leftrightarrow$

$10 \cdot 16^0 = 10$

$11 \cdot 16^1 = 176$

$6 \cdot 16^2 = 6 \cdot 16 \cdot 16$

$8 \cdot 16^3 = 8 \cdot 16 \cdot 16 \cdot 16$

$\Big\} \sum_{i=0}^{3} b_i \times 16^i$

$= 34490_{10}$

Hex Add:

carry: 1 1 0

$3BC5$ (15301_{10})

$+ \; 66E7$ (26343_{10})

$A2AC$ (41644_{10})

Hex Sub:

borrow: 1 1

$A2AC$ (41644_{10})

$- \; 66E7$ (26343_{10})

$3BC5$ (15301_{10})

Bin Add:

B	S
0+0	= 0 0
0+1	= 1 0
1+0	= 1 0
1+1	= 0 1
1+1+1	= 1 1

Bin Sub:

B	R
0-0	= 0 0
1-0	= 1 0
1-1	= 0 0
0-1	= 1 1

Unsigned Bin Sub: (borrows from nearest zero left bit)

01110^{16}

01000101

$-\;00110111$

00001110

Unsigned Bin Add:

carry: 1 1 1 1 1 1 1

$01000101 + 00111011 = 01111100$

Signed (V)

- 8-bit # range: -2^7 to $2^7-1 \rightarrow -128 \text{ — } +127$
- 16-bit # range: -2^{15} to $2^{15}-1 \rightarrow -32768 \text{ — } +32767$

Unsigned (C)

- 8-bit # range: 0 to $2^8-1 \rightarrow 0 \text{ — } 255$
- 16-bit # range: 0 to $2^{16}-1 \rightarrow 0 \text{ — } 65536$

$65536_{16} \equiv 8FFFF$

FLAGS: Z-flag: Zero; N-flag: Negative

C-flag is ONLY for unsigned addition/subtraction overflow

- 1: Invalid result (overflow) (+)
- 0: Valid result

V-flag is for signed overflow in addition/subtraction (±)

$V = C_7 \oplus C_6 \Rightarrow$ if $V = 0$ result is VALID

(C_7: carry out of MSB; C_6: carry into MSB)

* Adding a + # & − # is always valid result ⇒ cannot go out of bounds

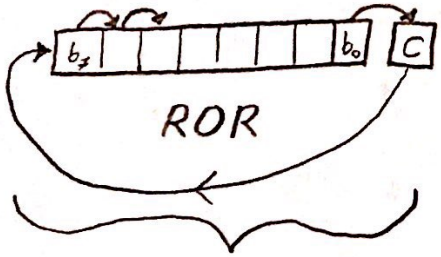
Rotate & Shift Instructions: (used to fasttrack multiplication)

Types (3) { (1) Rotate; (2) Logical Shift; (3) Arithmetic Shift } Left or Right

* (2) & (3) are register-level operations where a 1-bit shift left is equivalent to multiplying by 2

(1) Rotate: keep all content.

(After 9 rotations for a byte, original data will be back)

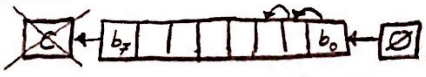


Right Rotation (C → b_7)

(Left Rotation is C → b_0) - ROL

(2) Logical: Discard carry value!

LSL (shift left) == ×2 (for 1-bit shift)



LSR (shift right) != ÷2 (DO NOT CONFUSE)

(3) Arithmetic:

ASL == ×2

ASR == ÷2

} for each 1-bit shift

ASL (shift left ⇔ LSL)

ASR (shift right — only useful for negative numbers)

Feed signed bit (MSB) back into the register

Scanned with CamScanner