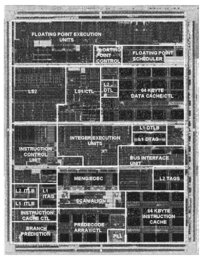


ITKC11 Mobile application platforms

http://tisu.mit.jyu.fi/embedded/ITKC11/ITKC11.htm

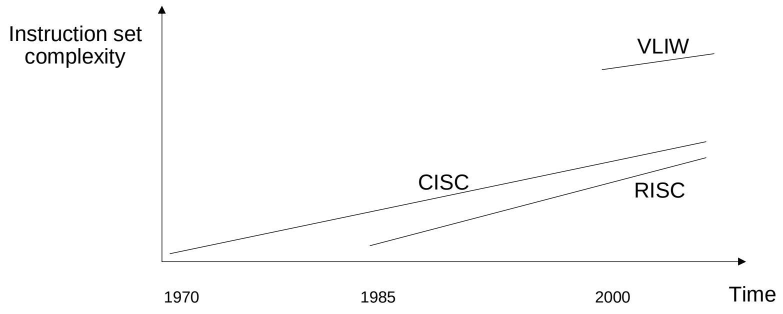
Processor architectures

Ville Pietikäinen and Jarkko Vuori

(Jarkko.Vuori@jyu.fi)

Processor architectures

- Chronologically CISC, RISC, VLIW



CISC

- CISC
- Complex Instruction Set Computer

- Typically low amount of registers
- Complex instructions, one instruction handles relatively large amount of operations or data
- Instruction word length varies between one and ten bytes
- Instructions are decoded and executed with microcode

- Processor / microcontroller examples: x86, H8S, 8051, MN102

CISC

- For example: Some code of Matshusita MN102-family processor
- 16-bit architecture, instructions still contain lot of words with odd length in bytes
- The longest instructions have length of 7 bytes and are very complex regarding functionality (TBZ, Test Bit Zero / Indirect memory addressing / specified bit / branch target address)

```
0000:4099: F4 44 99 04 43          MOVB    D0,(430499)      ;eeprom_65 = code
0000:409E: F3 FE C5 AA 89 00 0D    TBZ     (0089AA).5,40B2  ;eeprom_04.5 == 0?
0000:40A5: F4 C8 98 04 43          MOVBU   (430498),D0
0000:40AA: F5 08 01 00 00          OR      01,D0
0000:40AD: F4 44 98 04 43          MOVB    D0,(430498)      ;eeprom_64 |= 01
0000:40B2: F3 FE C1 B8 89 00 0D    TBZ     (0089B8).1,40C6  ;eeprom_12.1 == 0?
0000:40B9: F4 C8 98 04 43          MOVBU   (430498),D0
0000:40BE: F5 08 02 00 00          OR      02,D0
0000:40C1: F4 44 98 04 43          MOVB    D0,(430498)      ;eeprom_64 |= 02
0000:40C6: F4 C8 94 04 43          MOVBU   (430494),D0
0000:40CB: F7 48 DA 00 00          CMP     00DA,D0
0000:40CF: E9 1F 00 00 00          BNE     000040F0
```

RISC

- RISC
- Reduced Instruction Set Computer

- Processor architecture is simplified by reducing amount of instructions and their functionality
- Leads to small silicon area and thus low cost, for example, ARM7TDMI manufactured in 0.18 micron process takes about 0.53 mm^2 of silicon
- Fast, ARM7TDMI approximately 110 MHz @1.8V
- Pipelined → instructions per cycle average slightly larger than one instruction per clock cycle
- Typically have large register set

RISC

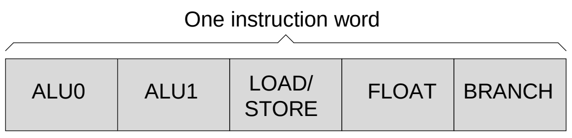
- For example: Some code of Fujitsu FR (MB91101) -processor
- 16/32-bit architecture, all instructions have the same length with exception being 32-bit Immediate Load (LDI:32) where the immediate value follows the instruction word
- Instruction word length even at 16 bits

```
001D:F366: 1781          ST       RP,@-R15
001D:F368: 0F01          ENTER    004h
001D:F36A: D79B          CALL     001DE2A2
001D:F36C: C154          LDI      #15h,R4
001D:F36E: C065          LDI      #06h,R5
001D:F370: 9F86          LDI:32   #112511DD,R6       ;112511DD
001D:F376: 9F8C          LDI:32   #00102BEE,R12      ;00102BEE
001D:F37C: 971C          CALL     @R12
001D:F37E: C154          LDI      #15h,R4
001D:F380: C055          LDI      #05h,R5
001D:F382: 9B06          LDI:20   #03E00,R6          ;3E00
001D:F386: 9F8C          LDI:32   #00102BEE,R12      ;00102BEE
001D:F38C: 971C          CALL     @R12
001D:F38E: 9F90          LEAVE
001D:F390: 0781          LD       @R15+,RP
001D:F392: 9720          RET
```

VLIW

- Very Long Instruction Word
- One instruction word has actually several instructions packaged together
- Instruction length up to 128 bits or even more
- No instruction decoder, the whole instruction word is fed to the execution unit as it is
- For example, well known Transmeta TM3200 is pure VLIW-processor and x86 compatibility has been reached with emulation

- Difficult target for C-compilers



Also in digital signal processors: TI's C6000

Processor architectures

- Differences between architectures have been getting smaller lately. Traditional CISC-processors have microarchitecturally changed to RISC-processors. Good example of this kind of trend are modern x86-family microprocessors.
- RISC-processors on the other hand have started to resemble CISC-processors in functionality. This is due to expanding instruction set and added instruction level complexity being designed in new RISC-processors. This can be seen for example in Fujitsu FR-series RISC-microcontrollers.
- VLIW is still young architecture but it can already be seen that parallelism is increasingly being applied also in CISC- and RISC-processors.

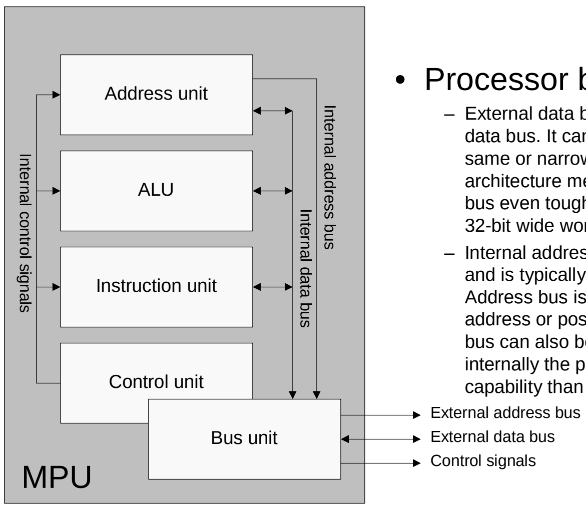
Embedded processors

- Microprocessors by definition are just processor cores with external bus interface.
- Microcontrollers are defined as integrated processor cores with peripheral functions.
- Single-chip microcontrollers are referred as microcontrollers with required memories integrated on chip.
- In embedded systems, all three types are used generally. Microprocessors have applications in single board computers that are essential part of larger systems. Microcontrollers are used in smaller standalone systems. Single-chip microcontrollers have their applications in smallest of the systems. However, the applications of all three type processors may overlap with each other depending on the application.

Processor structure

- A processor can be divided into basic functional units:
- Bus unit
  - Handles all the external data bus traffic of the processor. Interface out of the processor is data bus, address bus and a collection of control signals.
- Instruction unit
  - Fetches and decodes subsequent instructions of a program. The result of decoding is execution state defined by the instruction.
- ALU (Arithmetic-Logical Unit)
  - Execution unit that handled all arithmetic and logical operations.
- Addressing unit
  - Calculates the address that will be given to the bus unit.
- Control unit
  - Controls the internal operation of the processor according to information received from the instruction unit

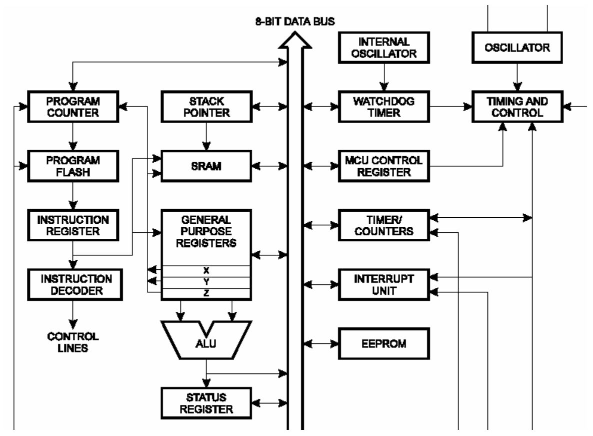
Processor structure



- Processor buses:
  - External data bus is practically extension of the internal data bus. It can have different width but typically is of same or narrower width. Typically a 16/32-bit architecture means processor with 16-bit external data bus even tough all internal operations are handled in 32-bit wide words.
  - Internal address bus is generated with addressing unit and is typically routed outside of the chip as it is. Address bus is used to select the desired memory address or possibly a I/O-address. Internal address bus can also be wider than external one meaning that internally the processor has more memory addressing capability than externally (virtual memory addressing).

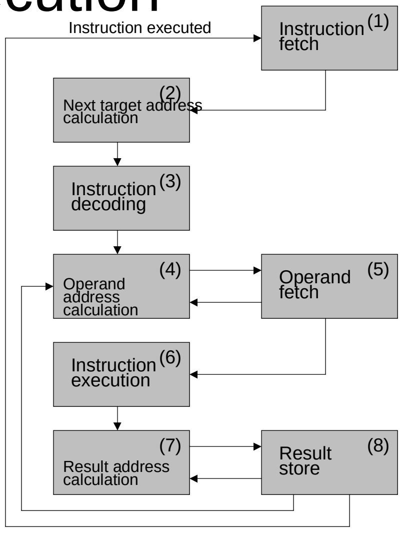
Processor structure

- The core of a simple microcontroller, Atmel AT90S2313:



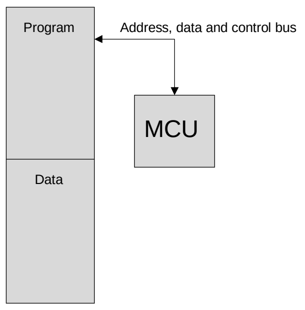
State diagram of instruction execution

- Events happening in one instruction cycle presented in state diagram
- Left side states are processor controlled
- Right side states are controlled by a memory
- First three states are instruction decoding, states 4 - 8 contain the instruction execution phase


Instruction executed
Instruction fetch (1)
(2) Next target address calculation
Instruction decoding (3)
(4) Operand address calculation
Operand fetch (5)
Instruction execution (6)
(7) Result address calculation
Result store (8)

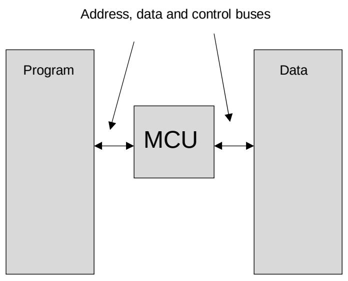
Address space architecture

- Memory is needed for programs and data
- Program memory used in embedded systems is typically of non-volatile type
- Data memory used is SRAM- or DRAM- type volatile memory



Von Neumann-architecture:

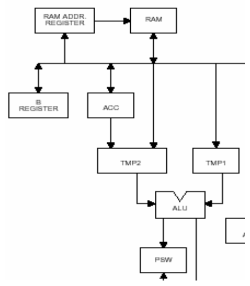
Program and data located in same address space



Harvard-architecture:
Separate address spaces and buses for program and data memories

Register structure: Accumulator based

- Accumulator architecture
- All arithmetic-logical operations are handled through accu-register
- Processor examples: Intel 8051, Motorola 68HC05, Zilog Z80 and generally all older types

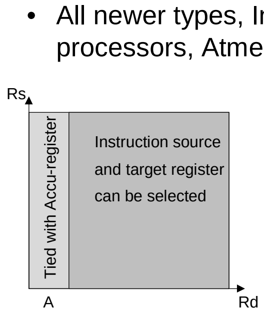


A piece of 8051-core

```
00A9 EC         mute:     mov a,r4           ;check mute status
00AA B40013               cjne a,#0,demute
00AD EA                   mov a,r2           ;save r2 and r3
00AE FD                   mov r5,a
00AF EB                   mov a,r3
00B0 FE                   mov r6,a
00B1 7AFF                 mov r2,#0ffh       ;mute left & right
00B3 7BFF                 mov r3,#0ffh
00B5 1200C9               lcall setvol
00B8 ED                   mov a,r5           ;restore r2 and r3
00B9 FA                   mov r2,a
00BA EE                   mov a,r6
00BB FB                   mov r3,a
00BC 7CFF                 mov r4,#0ffh       ;mute status = on
00BE 8005                 sjmp back
00C0 1200C9     demute:   lcall setvol
00C3 7C00                 mov r4,#0
00C5 22         back:     ret
```

Register structure: general

- Register-register architecture
- Operations can be made between all registers
- Register contents can be moved, loaded and stored freely
- Also known as orthogonal architecture
- All newer types, Intel 80x86, Hitachi H8/300, H8S, All RISC-processors, Atmel AVR, Motorola 68HC12



```
0002:AB2A: 914F     MOV.W    @(4F,PC),R1    ;0002ABCC = 008B
0002:AB2C: 3910     CMP/EQ   R1,R9          ;compare to byte 8B
0002:AB2E: 8B22     BF       22:8           ;0002AB76 return with 1 in R0
0002:AB30: 4A0B     JSR      @R10           ;call 6Fxxx
0002:AB32: E402     MOV      #02:8,R4       ;region 2?
0002:AB34: 2008     TST      R0,R0
0002:AB36: 8902     BT       02:8           ;0002AB3E was zero
0002:AB38: E306     MOV      #06:8,R3
0002:AB3A: 3B37     CMP/GT   R3,R11         ;R11 = region
0002:AB3C: 8B07     BF       07:8
```

What about stack architecture???

Micro architecture

- Instruction set defines the processor *architecture*
- General processor architecture is also known as Instruction Set Architecture (ISA)

- *Microarchitecture* tells how the instruction set is executed

- Life span of a same instruction set realizing microarchitecture is typically about 6 years. Renewal cycle is thus quite slow.
- During this time, performance of microcontrollers increase mostly due to development in integration processes.

Increasing execution speed

- In addition to increasing clock frequency several architectonic improvements can be applied:
  – Pipelining, super pipelining
  – Superscalarity
  – Out of order execution
  – Register renaming
  – Speculative execution
  – Partly decoded instruction storage to separate cache memory
  – Caching
- These methods are widely used in modern microprocessors. Slowly but surely they are finding their way to embedded processor cores.

Pipeline

- Instruction execution is split in parts
- For example, Fetch, Decode, Execute, Write

Fetch	Decode	Execute	Write		
	Fetch	Decode	Execute	Write	
		Fetch	Decode	Execute	Write

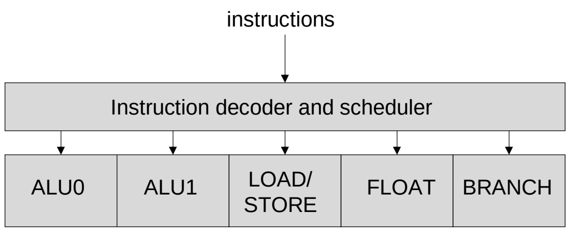
- Inevitably pipelining is disturbed when instructions have dependencies with each other. This leads to flushing the pipeline and starting execution for relevant instructions again. Flushing leads to momentary decrease in performance.

Pipeline

- Pipeline length: ARM7: 3 stages, compare to P4: 20 stages
- Latency = time for one instruction to pass through the pipeline. Practically the same time for non-pipelined execution.
- Throughput = Effect of the pipeline to instruction execution time. For example, if an instruction execution takes 12 clock cycles without pipeline and only one clock cycle with pipeline is throughput with a pipelined processor core 12 times compared to the non-pipelined version.

Super scalar execution

- Superscalarity means using several execution units
- Instructions are issued for different execution units and thus instruction level parallelism is added inside the processor core.
- Instruction scheduler issues the instructions and tries to keep each execution unit busy. This leads to maximal usage of available resources.



Out of order execution

- Instructions received from the instruction decoder are issued between execution units in scheduler.
- Issuing tries to schedule the instructions so that each execution unit is utilized as much as possible.
- → Added parallelism is properly taken advantage of
- After execution the instruction deliverables are regrouped together to check for dependencies
- Write back is handled according to instruction level dependencies

Register renaming

- A part of out of order execution strategy
- Processor core contains internal register more than is available to instruction set
- Processor core selects temporary registers to be used for the instruction being executed
- Is needed because out of order execution engines may need more registers than is available normally for the instruction set

Speculative execution

- Processor decodes instructions by ‘guessing’
- Has significance particularly with branching instructions

- In practice the processor core keeps a table of the most used branch target addresses. When branch instruction is being executed the processor can access the table and see what the probability for the branch being taken is statistically and move on to generate address for the branch target address. Most probable branch target address data is probably already in cache memory and thus accessible very fast.
- The branch result not being what was speculated leads to momentary penalty in performance as new branch target address needs to be calculated.

Execution trace cache

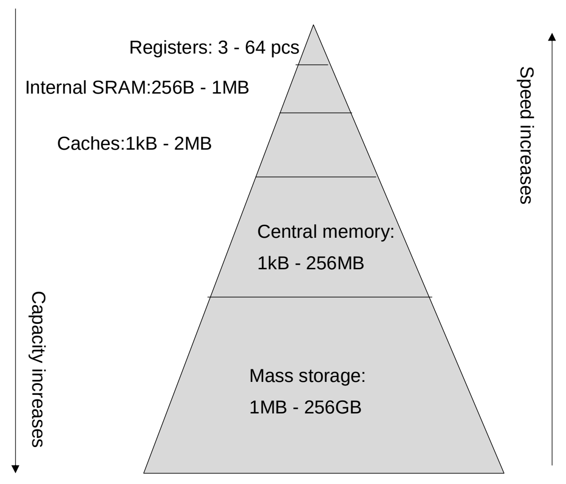
- Means separate cache memory for decoded instructions
- CISC-processor instructions are split into so called micro-operations in modern microprocessors. Execution trace cache means storing these micro-operation groups describing one CISC-instruction to separate cache memory area for fast access when required again.
- This leads to saving execution time as the instructions are not necessary to decode again each time they are used.
- First used in P4.
- Execution trace cache is quite limited in capacity, for example in P4 20k micro-operations.

Memory hierarchy

Interfacing processor and memories...

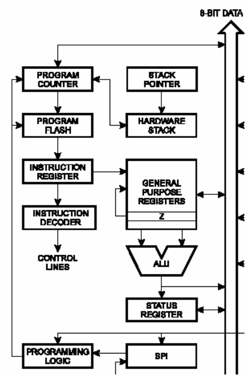
Memory hierarchy

- Memory hierarchy:
  - 1. Registers
  - 2. Internal SRAM
  - 3. Caches
  - 4. Central memory
  - 5. Mass storage, virtual memory



Registers

- Smallest of the systems work without internal SRAM-memory for data. This means that registers are the only data memory available to the program.
- For example, AT90S1200:



Processor core does not contain SRAM:

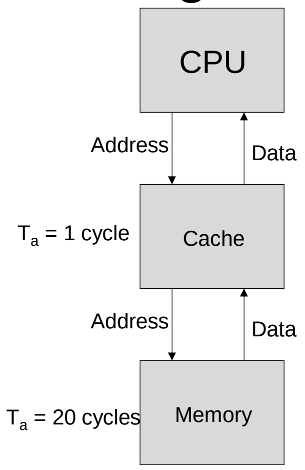
- Only registers available for data storage
- Half of the registers immediately available
- 3-level hardware stack
- No external program or data memory
- No addressing capability for external memory
- Has however 64 bytes of EEPROM-memory

Cache

- Cache means very fast temporary memory between processor core and central memory. Some embedded processors have typically 1 - 16 kilobytes of cache memory.
- Due to speed requirement cache is of SRAM-type and typically integrated on the same die with the processor core
- Cache can have different hierarchy levels: first, second and even third level caches exist. Embedded processors use typically only one level of cache memory.
- Cache memory can be divided between instructions and data or both can have separate caches. This depends on if the processor architechture is of von-neumann or harvard type.
- Caches have several structures: direct mapped, set associative and fully associative.

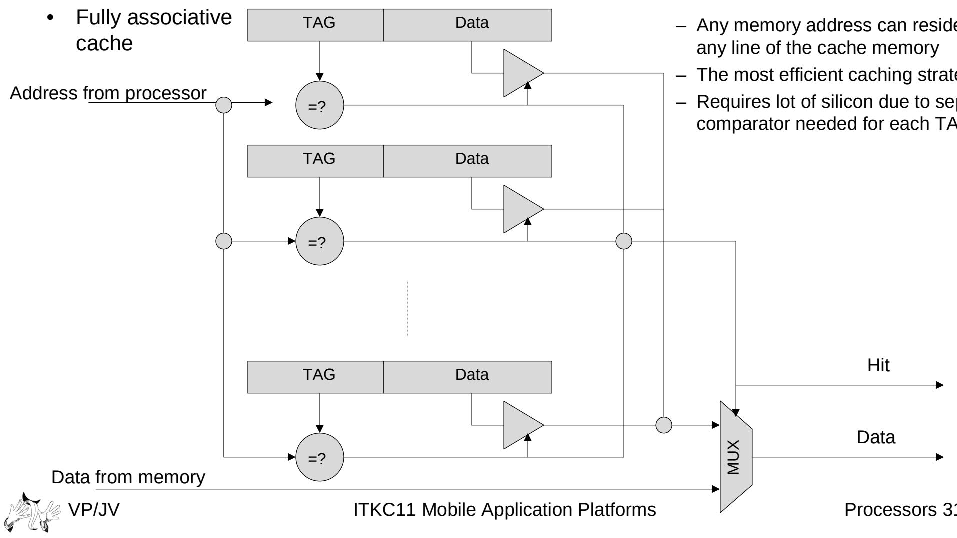
Cache advantage

- For example: Let us assume that data can be read from cache in one clock cycle (t_c). Ordinary memory access (t_m) on the other hand takes 20 clock cycles.
- If cache hit accuracy is 95 percent (H), average access time seen by the processor core (t_a) is less than two clock cycles (1.95 to be exact). Compare this with the 20 cycles for ordinary access to memory.
  – $t_a = Ht_c + (1-H)t_m$



Cache memory

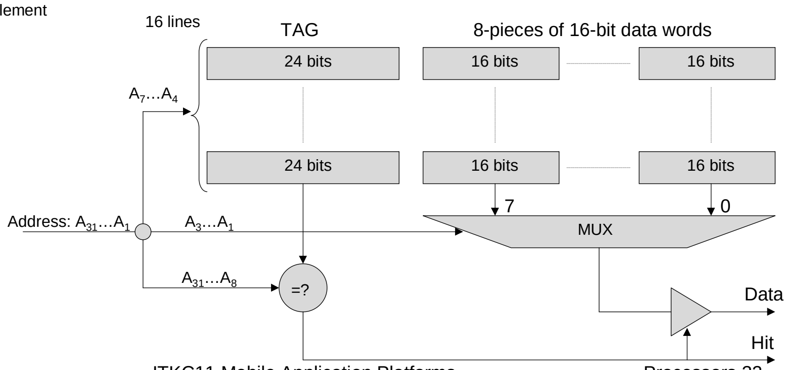
- Fully associative cache



- Any memory address can reside in any line of the cache memory
- The most efficient caching strategy
- Requires lot of silicon due to separate comparator needed for each TAG

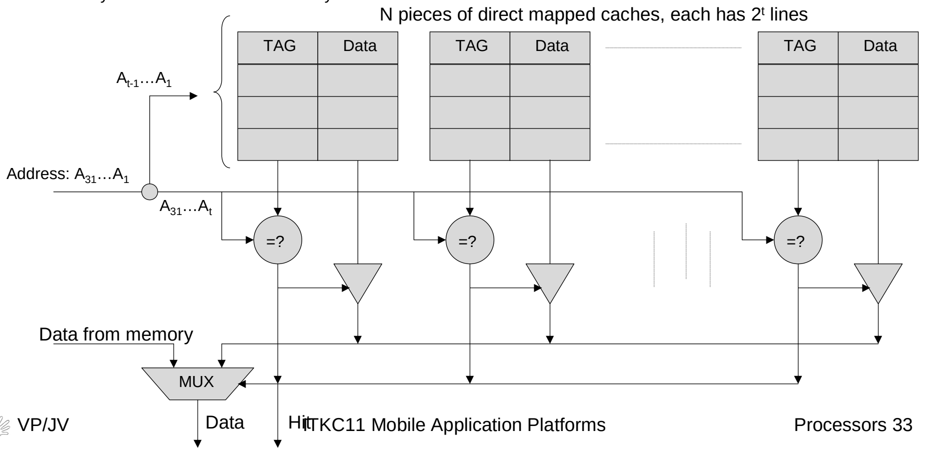
Cache memory

- Direct mapped cache:
- Each memory address is able to reside in only one line
  - TAG requires lot of SRAM, only one comparator needed
  - Does not need replacement strategy
  - Easy to implement



Cache memory

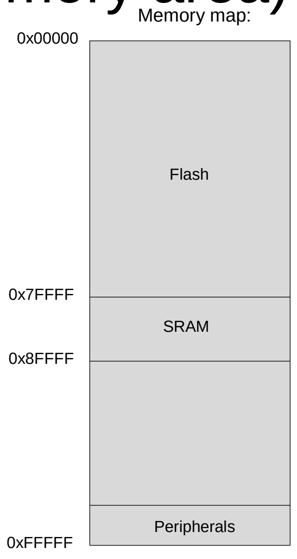
- N-way set-associative cache:
  - Combination of fully associative and direct mapped cache
  - 8-way SAC almost as efficient as fully associative in same size



N pieces of direct mapped caches, each has 2^t lines
TAG
Data
$A_{t-1} \ldots A_1$
Address: $A_{31} \ldots A_1$
$A_{31} \ldots A_t$
=?
Data from memory
MUX
Data
Hit

Address space (memory area)

- Program memory is typically non-volatile flash or EPROM memory
- Can contain also SRAM- or DRAM- type volatile memory for data use
- All embedded systems contain some amount of non-volatile memory for storing the program code. A processor cannot function unless it has instructions to execute after reset.



Instruction set of a processor

General instruction set, addressing modes...

Instruction set

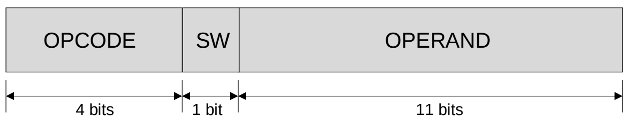
- Processor architecture is defined by it's instruction set (ISA)
- Generally instruction set contains a number of basic instructions. Differences between architectures come from processor specific special instructions.
- Instruction set is highly dependant on the processor structure (CISC, RISC or whatever).

- Essential part in defining instruction set is the register set of the processor. In addition to integer register, also floating point registers can be available in a processor.
- Special registers are for example program counter (PC), Program Status register (PS, Flags) and stack pointer (SP).

Instruction format

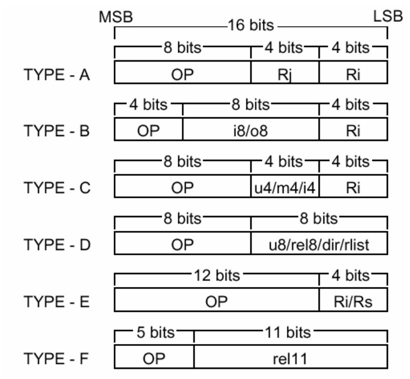
- Instruction format tells how the instructions are coded in binary
- Instruction word contains usually the instruction itself and operand or several of them. In addition to these the instruction word can contain conditioning and flag codes to modify instruction execution.

The format of an imaginary 6-bit instruction word:



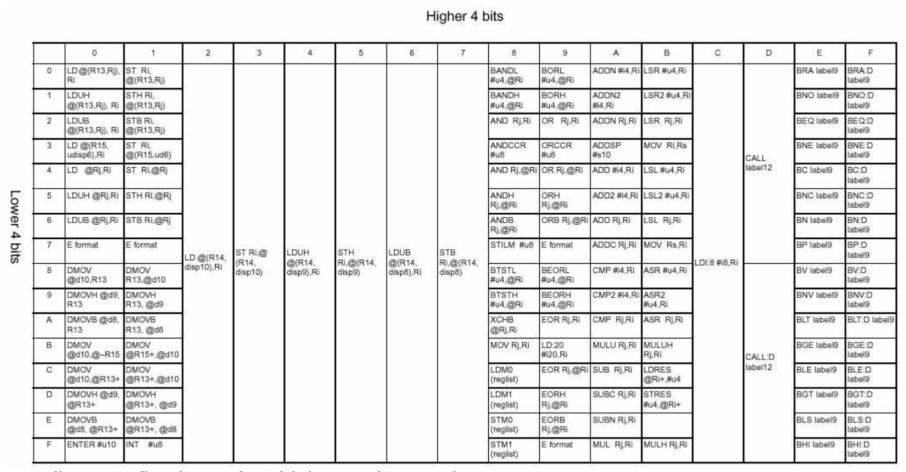
Instruction format

- Without exceptions, the instructions come in different formats.
- Instruction type information (OPCODE) can vary quite a lot between different instruction words.



Instruction format

- Instruction map:



Higher 4 bits

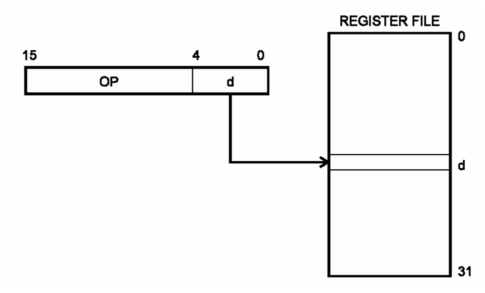
Lower 4 bits	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	LD @(R13,Rj), Ri	ST Ri, @(R13,Rj)							BANDL #u4,@Ri	BORL #u4,@Ri	ADDN #i4,Ri	LSR #u4,Ri			BRA label9	BRA:D label9
1	LDUH @(R13,Rj), Ri	STH Ri, @(R13,Rj)							BANDH #u4,@Ri	BORH #u4,@Ri	ADDN2 #i4,Ri	LSR2 #u4,Ri			BNO label9	BNO:D label9
2	LDUB @(R13,Rj), Ri	STB Ri, @(R13,Rj)							AND Rj,Ri	OR Rj,Ri	ADDN Rj,Ri	LSR Rj,Ri			BEQ label9	BEQ:D label9
3	LD @(R15, udisp6),Ri	ST Ri, @(R15,ud6)							ANDCCR #u8	ORCCR #u8	ADDSP #s10	MOV Ri,Rs		CALL label12	BNE label9	BNE:D label9
4	LD @Rj,Ri	ST Ri,@Rj							AND Rj,@Ri	OR Rj,@Ri	ADD #i4,Ri	LSL #u4,Ri			BC label9	BC:D label9
5	LDUH @Rj,Ri	STH Ri,@Rj							ANDH Rj,@Ri	ORH Rj,@Ri	ADD2 #i4,Ri	LSL2 #u4,Ri			BNC label9	BNC:D label9
6	LDUB @Rj,Ri	STB Ri,@Rj							ANDB Rj,@Ri	ORB Rj,@Ri	ADD Rj,Ri	LSL Rj,Ri			BN label9	BN:D label9
7	E format	E format	LD @(R14, disp10),Ri	ST Ri,@ (R14, disp10)	LDUH @(R14, disp9),Ri	STH Ri,@(R14, disp9)	LDUB @(R14, disp8),Ri	STB Ri,@(R14, disp8)	STILM #u8	E format	ADDC Rj,Ri	MOV Rs,Ri	LDI:8 #i8,Ri		BP label9	BP:D label9
8	DMOV @d10,R13	DMOV R13,@d10							BTSTL #u4,@Ri	BEORL #u4,@Ri	CMP #i4,Ri	ASR #u4,Ri			BV label9	BV:D label9
9	DMOVH @d9, R13	DMOVH R13, @d9							BTSTH #u4,@Ri	BEORH #u4,@Ri	CMP2 #i4,Ri	ASR2 #u4,Ri			BNV label9	BNV:D label9
A	DMOVB @d8, R13	DMOVB R13, @d8							XCHB @Rj,Ri	EOR Rj,Ri	CMP Rj,Ri	ASR Rj,Ri			BLT label9	BLT:D label9
B	DMOV @d10,@–R15	DMOV @R15+,@d10							MOV Rj,Ri	LD:20 #i20,Ri	MULU Rj,Ri	MULUH Rj,Ri		CALL:D label12	BGE label9	BGE:D label9
C	DMOV @d10,@R13+	DMOV @R13+,@d10							LDM0 (reglist)	EOR Rj,@Ri	SUB Rj,Ri	LDRES @Ri+,#u4			BLE label9	BLE:D label9
D	DMOVH @d9, @R13+	DMOVH @R13+, @d9							LDM1 (reglist)	EORH Rj,@Ri	SUBC Rj,Ri	STRES #u4,@Ri+			BGT label9	BGT:D label9
E	DMOVB @d8, @R13+	DMOVB @R13+, @d8							STM0 (reglist)	EORB Rj,@Ri	SUBN Rj,Ri				BLS label9	BLS:D label9
F	ENTER #u10	INT #u8							STM1 (reglist)	E format	MUL Rj,Ri	MULH Rj,Ri			BHI label9	BHI:D label9

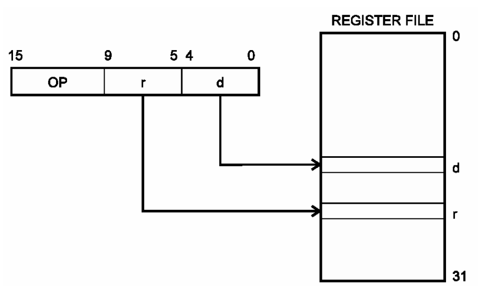
Fujitsu FR, first byte of 16-bit instruction word

Addressing modes

- One register direct addressing
- Two register direct addressing

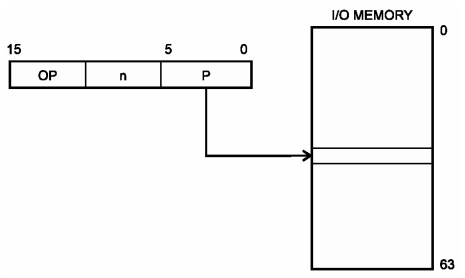
Atmel AVR AT90S2313 supported addressing modes

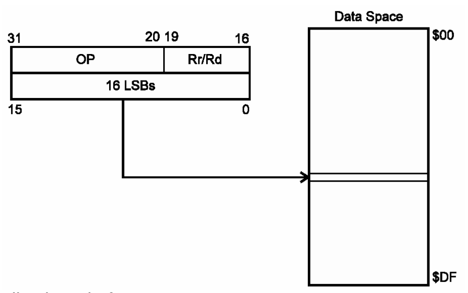




Addressing modes

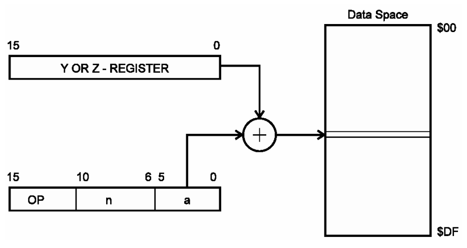
- Direct I/O addressing
- Direct data addressing (this is harvard-type microcontroller)

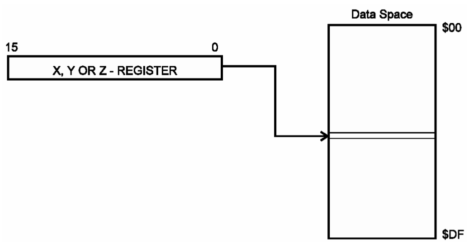




Addressing modes

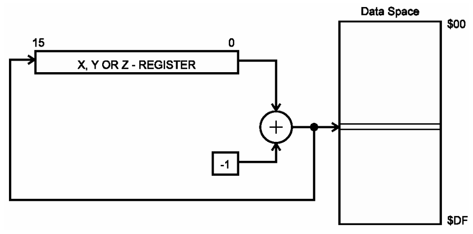
- Indirect data addressing with offset (index register)
- Indirect data addressing with only index register



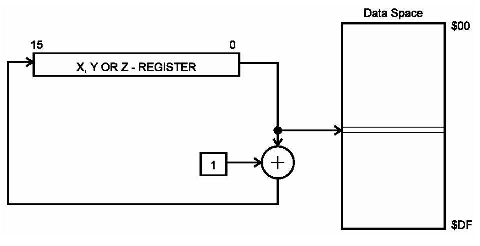


Addressing modes

- Indirect data addressing with pre-decrement

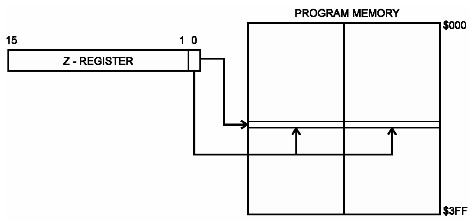


- Indirect data addressing with post increment

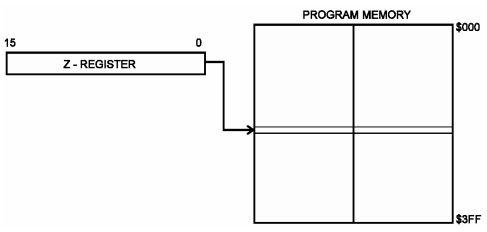


Addressing modes

- Addressing to program memory for constant fetch

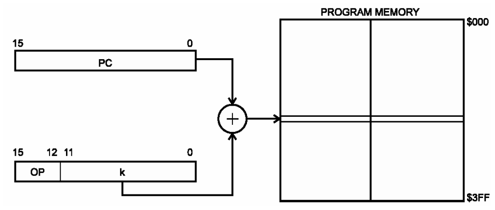


- Indirect program memory addressing for jump instructions



Addressing modes

- Relative program memory addressing for branch instructions



Basic instructions

- The instructions can be categorized according to type:

- Data transfer (MOV, LD, ST, PUSH, POP, SWAP)
- Arithmetic (ADD, SUB, MUL, MAC, DIV, INC, DEC, NEG, EXTS, SAL, SAR)
- Shift /rotation (SHL, SHR, ROTL, ROTR)
- Bit manipulation (AND, OR, XOR, NOT, BSET, BCLR)
- Test and comparison (CMP, TEST)
- Branching (conditional, JSR, JMP, RET, RETI)
- Also processor dependant instructions exist for controlling the operation of processor and peripherals.

Data transfer

- The simplest data transfer operation is to copy data from a location to another location.
- MOVe, move from a location to another
- LoaD, load from memory to register
- STore, store from register to memory
- SWAP, changes two storage location contents with each other. Locations can be two registers or a register and memory location. Register internals can also be swapped, for example, MSB and LSB of a 16-bit register.
- PUSH, store to stack (stack pointer modified)
- POP, fetch from stack (stack pointer modified)

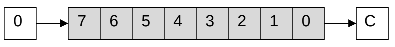
Arithmetic

- ADD means addition of two operands. Can also include carry flag when the syntax would be ADDC. F.ex, ADDC R1, R2 equals R1 = R1+R2+C
- SUB means substraction of two operands. Can also include carry-flash as in ADD-operation. F.ex, SUBC R1,R2 equals R1 = R1-R2-C
- MUL means multiplication of two operands. Operand length in MUL-operation can vary depending on the processor architecture. Typical lengths are 8 x 8, 16 x 8, 16 x 16, 32 x 8 and 32 x 16. Multiplication can include sign (MULU, MULS, MULXU etc).
- DIV means division of a operand with another one. Operand length can vary as in multiplicatoin. Division can also be signed (DIVU, DIVS).

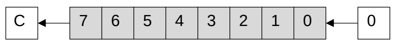
Arithmetic

- MAC stands for multiplication and subsequent addition (Multiply ACcumulate). First a two operand multiplication is performed (like in ordinary MUL) and the result is summed with a third operand.
- INC increases the operand value with one. INC R1 equals thus with instruction ADD R1,1 (R1 = R1 + 1).
- DEC decreases the operand value with one. F.ex, DEC (R1) would substract one from the contents of address R1.
- NEG is direct negation and changes the sign of the operand.
- EXTS means extension of sign. For example, an 8-bit number can be cast to 16-bit with EXTS instruction.
- SAL, SAR (or ASL, ASR) are arithmetic shift instructions. Instructions preserve the sign of the original operand.

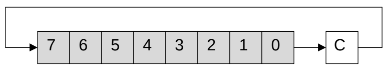
Shift / Rotate



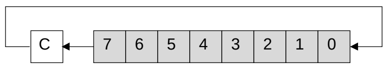
- SHR, LSR ………...



- SHL, LSL ………...



- ROTR …………………...



- ROTL …………………...

Bit manipulation

- AND as logical operation makes possible to compare two operands and save bits that equal one.
- OR as logical operation makes possible to merge two operands with each other.
- XOR as logical exclusive or makes possible to change bit values in operand. Notice that f.ex XOR R1,R1 would zero out register R1.
- NOT as logical negation turns bit values in operand.
- BSET makes possible to set bit values in operand.
- BCLR makes possible to clear bit values in operand.
  - BSET and BCLR are very useful instructions to control embedded processor peripherals.

Comparison and testing

- Comparison
- CMP as in CoMPare is practically substraction with no result saving. For example, CMP R1,#05 substracts value 5 from the contents of register R1. If the result is zero, Z-flag is set. If the result is negative, O-flag is set (overflow).

- Testing
- TEST is practically an AND-instruction with no result saving. For example, TEST R1,#40 ‘tests’ if register R1 has it's sixth bit set as one.

Branching

- Direct jump, JuMP or BRAnch, is an unconditional jump instruction. The address given in operand is transferred to program counter and program execution moves to the desired address.
- Subroutine call, JSR, CALL or BSR is an unconditional branch to subroutine. Program counter contents is stored on stack and new value is received from operand. Program execution moves to the desired address.
- Return from subroutine, RET or RTS, fetches the program counter value from stack and tansfer thus program execution to the address the subroutine was called from. Similar instruction RETI means instruction which returns from an interrupt routine and is equivalent to RET except that the contents of status (flag) register before interrupt can be pulled out of stack.

Conditional branching

- Branching is conditioned with the contents of status register, usually after arithmetic instructions
- Z = Zero, C = Carry, N = Negative, V = oVerflow:

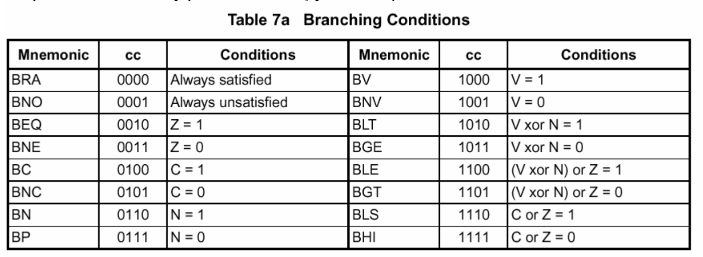


Table 7a Branching Conditions

Mnemonic	cc	Conditions	Mnemonic	cc	Conditions
BRA	0000	Always satisfied	BV	1000	V = 1
BNO	0001	Always unsatisfied	BNV	1001	V = 0
BEQ	0010	Z = 1	BLT	1010	V xor N = 1
BNE	0011	Z = 0	BGE	1011	V xor N = 0
BC	0100	C = 1	BLE	1100	(V xor N) or Z = 1
BNC	0101	C = 0	BGT	1101	(V xor N) or Z = 0
BN	0110	N = 1	BLS	1110	C or Z = 1
BP	0111	N = 0	BHI	1111	C or Z = 0

8051: CJNE == Compare Jump if Not Equal...

Real example 1

Low-pass filter block pointer by R0 using
Motorola DSP56001 24-bit Harward architecture
Based signal processor

```
530               ; then filter the left channel
531 P:006E 05F420 move #buflen*4-1,m0
           0003FF
532 P:0070 70F400 move #-4,n0
           FFFFFC
533
534               ; calculate even phase
535 P:0072 669B00 move           x:<eadr,r6
536 P:0073 225000 move r2,r0
537 P:0074 200013 clr  a
538 P:0075 F0C800 move           x:(r0)+n0,x0  y:(r6)+,y0
539 P:0076 061BA0 rep  #iftaps-1
540 P:0077 F0C8D2 mac  x0,y0,a   x:(r0)+n0,x0  y:(r6)+,y0
541 P:0078 61F4D3 macr x0,y0,a   #ocoeffs,r1
           000020
```

Address
Line number Program code Source code

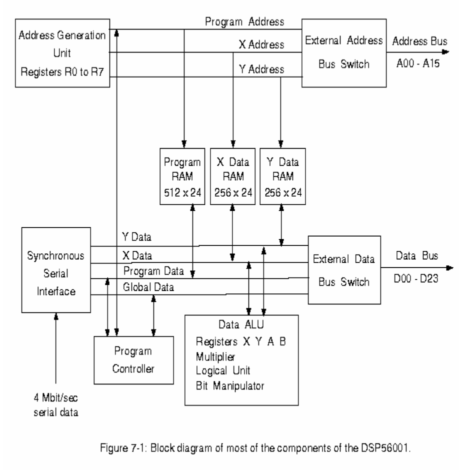
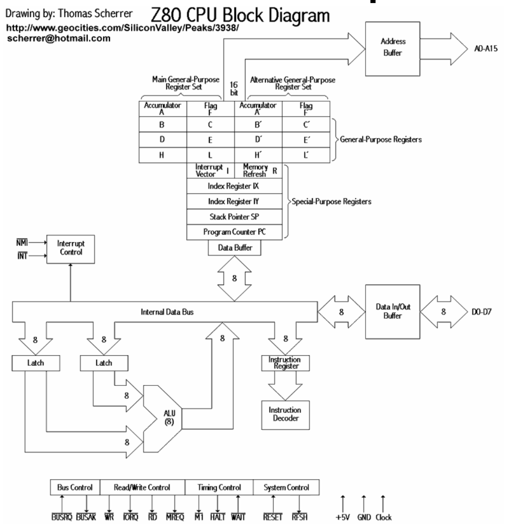


Figure 7-1: Block diagram of most of the components of the DSP56001.

Real example 2

Accumulator architecture
Z80 processor based light
Controller start-up sequence

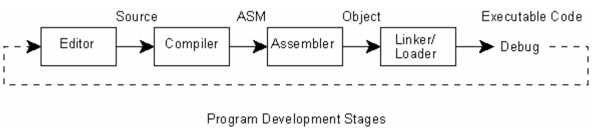
```
 92:           ;boot the system up
 93:  0018 310018 BOOT: LD  SP,RAM+RAMLEN
 94:            ;Romcheck
 95:  001B 210000       LD   HL,ROM1
 96:  001E 110008       LD   DE,ROM1+ROMLEN
 97:  0021 CD9B03       CALL CHKROM
 98:  0024 DACE03       JP   C,HWERR
 99:           ;
100: 0027 210008       LD   HL,ROM2
101: 002A 110010       LD   DE,ROM2+ROMLEN
102: 002D CD9B03       CALL CHKROM
103: 0030 DACE03       JP   C,HWERR
104:            ;Ramcheck
105: 0033 210010       LD   HL,RAM
106: 0036 110018       LD   DE,RAM+RAMLEN
107: 0039 CD8603       CALL CHKRAM
108: 003C DACE03       JP   C,HWERR
109:            ;Initialize I/O system,
110: 003F CDD103       CALL HWDINI
111:            ;multitasking kernel,
112: 0042 21BC10       LD   HL,FRAMES
113: 0045 018000       LD   BC,FRMLEN
114: 0048 3E08         LD   A,FRMCNT
115: 004A CD6800       CALL KERINI
116:            ;data - areas and
117: 004D CDF203       CALL DATINI
118:            ;interrupt system
119: 0050 ED5E         IM   2
120: 0052 3E07         LD   A,VECPG SHR 8
121: 0054 ED47         LD   I,A
122: 0056 FB           EI
123:            ;then jump to the main program
124: 0057  C30008      JP   ROM2
```



Real example 3

```
•                               ; void main(void) {
•                               ;     /* alusta ensin HC:n I/O-portit */
•                               ;     DDRA   = 0xFF;
•    85C8 C6FF     ldab #255
•    85CA F71001   stab  4097
•    85CD          simple.20:: ;     DDRD   = 0xFF; PORTD  = 0x00;
•    85CD C6FF     ldab #255
•    85CF F71009   stab  4105
•    85D2 4F       clra
•    85D3 5F       clrb
•    85D4 F71008   stab  4104
•    85D7          simple.21:: ;     DDRG  |= 0x0F;
•    85D7 F61003   ldab  4099
•    85DA CA0F     orab #15
•    85DC F71003   stab  4099
•    85DF        L4:
•    85DF        L5:
•    85DF          simple.25:: ;
•                               ;     /* vilkuttele LEDiä */
•                               ;     while (FOREVER) {
•                               ;             PORTD ^= bit2;
•    85DF F61008   ldab  4104
•    85E2 C804     eorb #4
•    85E4 F71008   stab  4104
•    85E7          simple.27:: ;
•                               ;             DELAY(30);
•    85E7 CC001E   ldd #30
•    85EA BD0000   jsr  _DELAY
•    85ED 30       tsx
•    85EE 20EF     bra  L4
•    85F0        L3:
•                               ;     }
•                               ; }
```

C-programing for the
Motorola 68HC11 microcontroller,
Simple LED blinking code using
multitasking kernel


Source
ASM
Object
Executable Code
Editor
Compiler
Assembler
Linker/ Loader
Debug
Program Development Stages