STM8S Core & Architecture

STM8S Technical Training

CONTENTS

- **Introduction to STM8**
  - Architecture
  - Instruction set
  - Performance
  - Comparison to ST7
- **New debug capabilities**
- **Interrupt controller**
- **STM8 Benefits**
- **Program and Data Memory**

Introduction

- The goal is to present you the new 8-bit core made in ST which will replace the ST7
- We will focus on the improvements of the STM8 versus the ST7

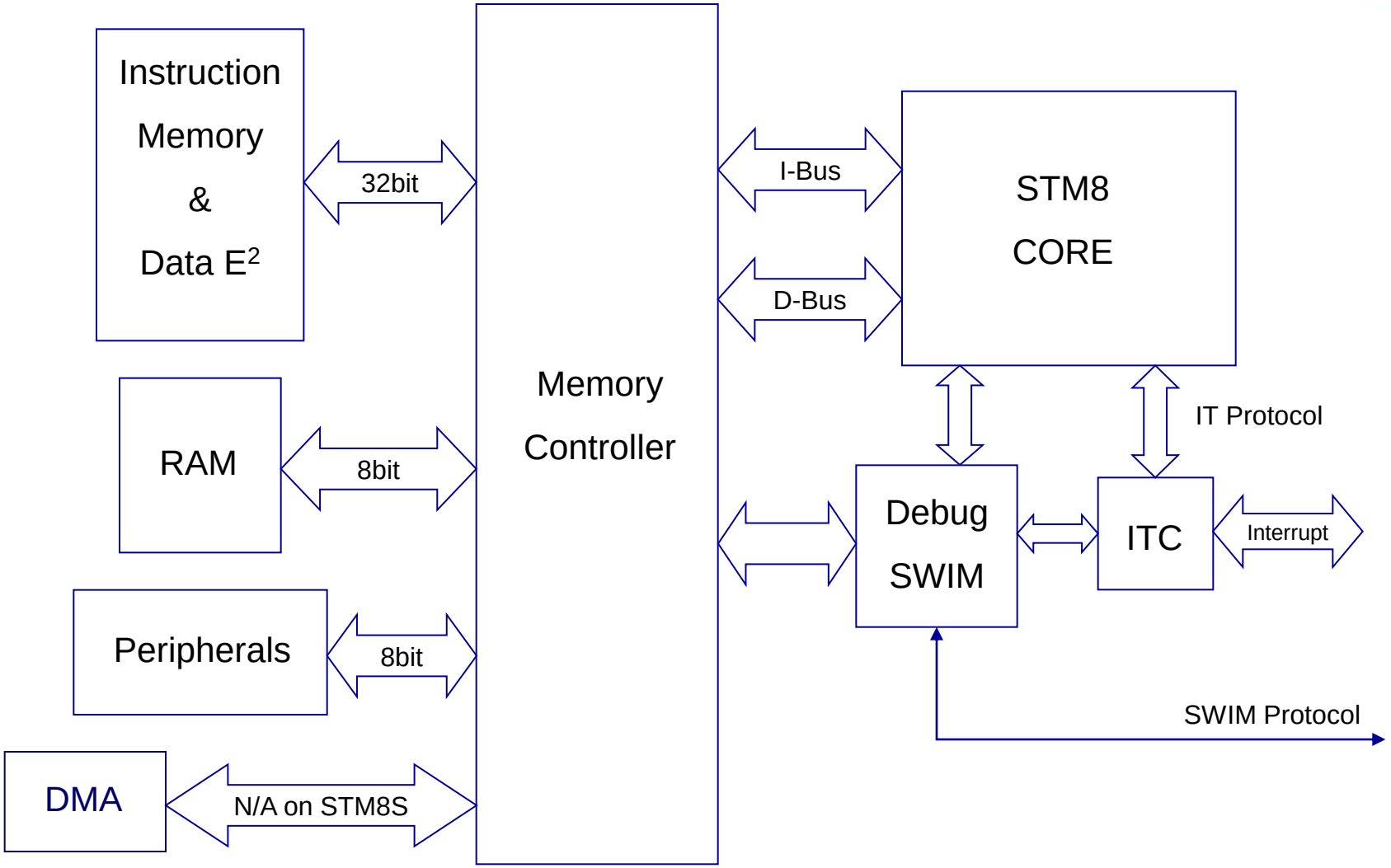
Architecture (1/2)

- **Harvard architecture**
  - 3-stage pipeline
  - CISC machine
  - Unified memory space
  - 32-bit program memory (EE/Flash) access
    - Pre-code only affects code density, not performance
  - 8-bit data access
    - Fast access to variables
    - Tightly connected RAM (register file extension)
  - Reduced CPI >> reduced clock frequency
    - Most of instructions executed in one cycle per instruction
  - Signed arithmetic operation support
  - Higher power efficiency

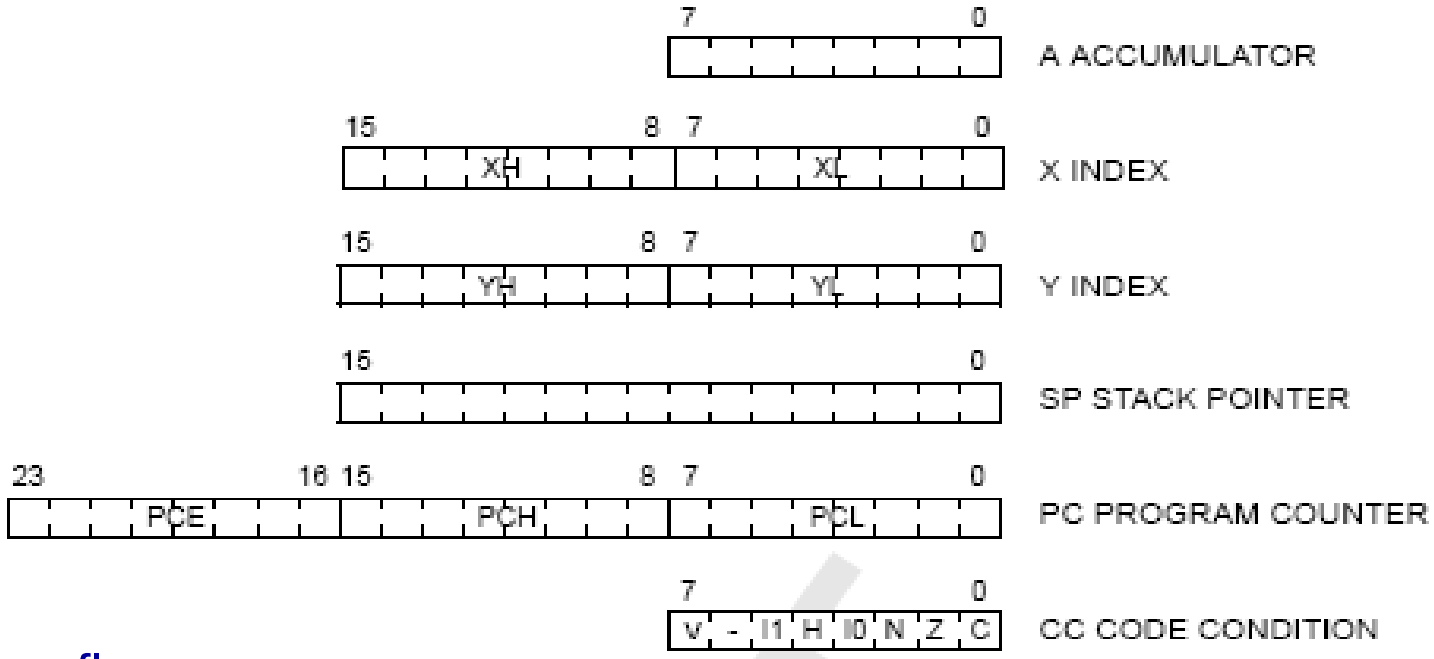
Architecture (2/2)

- **16Mbytes addressing space**
  - With a 24-bit Program Counter
- **16-bit index registers**
  - Remove all the limitations related to 0-page (1st 256 bytes of RAM)
    - Fast and bit accesses to whole RAM and register space
  - All indexed addressing modes will take advantage of full access to the 1st 64k segment access
  - Whole 0-page space available for global variables & literal pool
  - Benefits for Compiler in combination with new instruction set
    - More compact code for addressing tables in the whole 64k segment (8-bit offset only)
      - (off.b, X) addressing mode becomes much more powerful
    - Allows the usage of large arrays (>256 elements)
    - More efficient parameter passing onto the stack (using either SP or X/Y as frame pointer)
      - Standard memory model
      - Reentrance possible

STM8 System Implementation

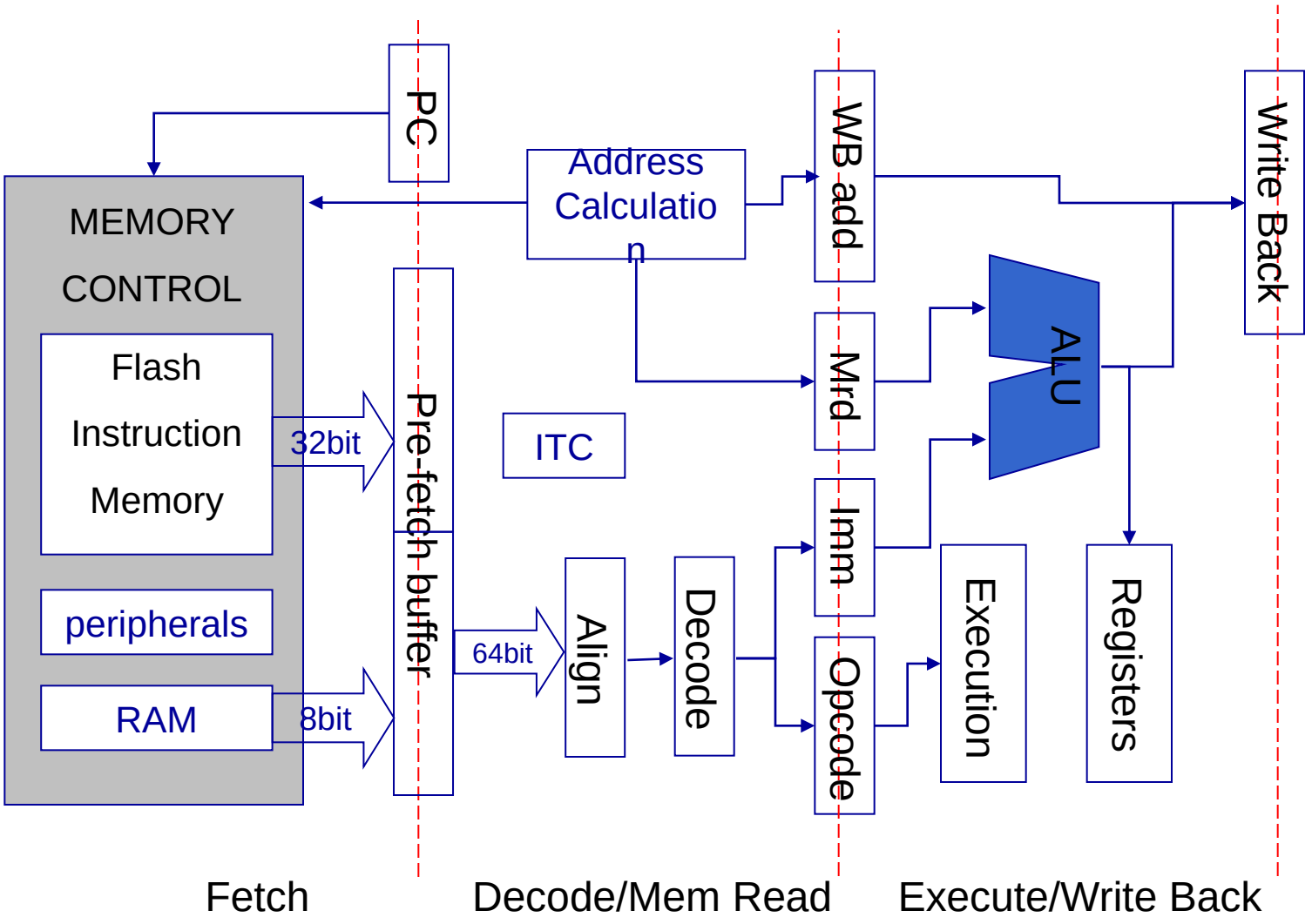


STM8 Core Register Set



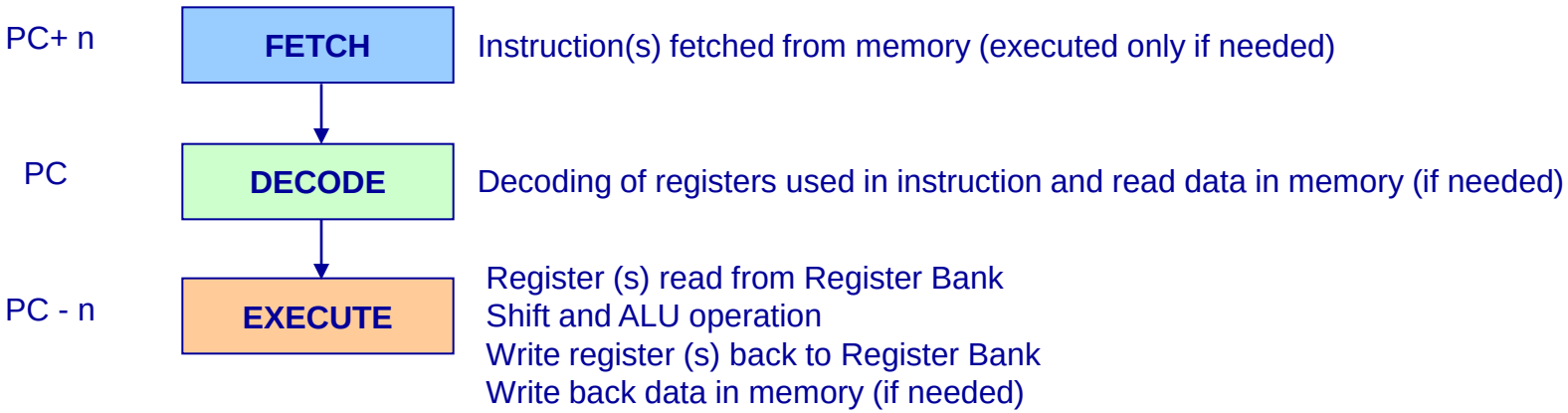
V: Overflow
H: Half-carry
N: Negative
Z: Zero
C: Carry
IO, I1: interrupt mask level 0, 1

CPU basic pipeline



The Instruction Pipeline

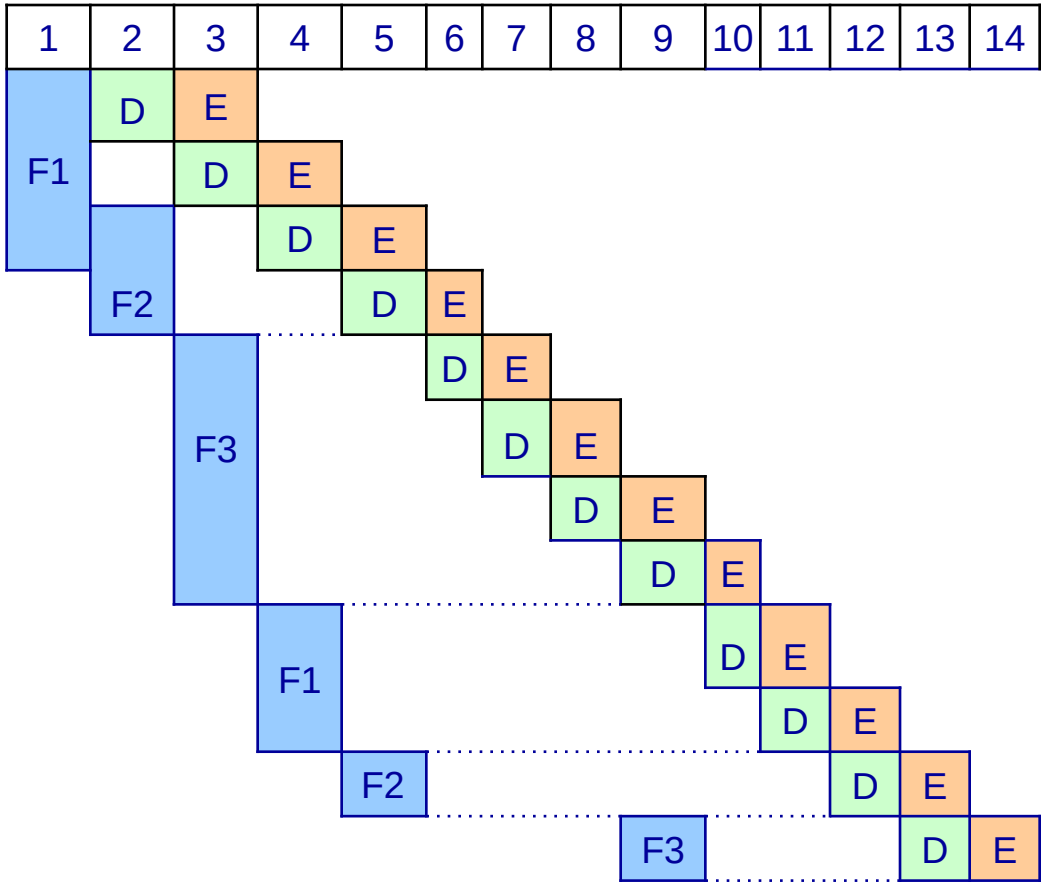
- The STM8 family uses a 3 stage pipeline in order to increase the speed of the flow of instructions to the processor.
  - Allows several operations to be undertaken simultaneously, rather than serially.



  - The PC points the instruction being decoded.

Optimal pipeline example

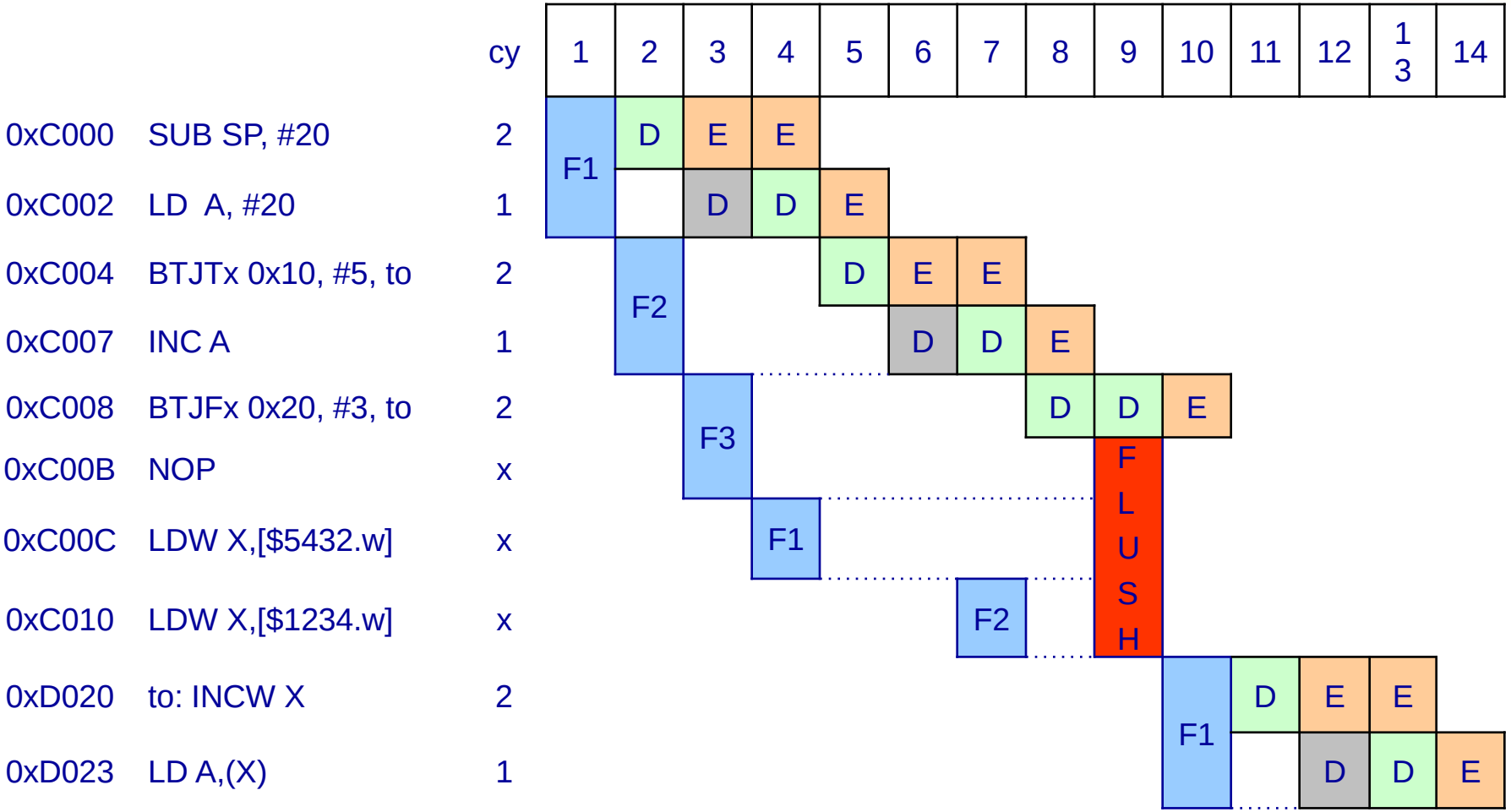
		cy	lg
0xC000	NEG A	1	1
0xC001	XOR A, $10	1	2
0xC003	LD X, #20	1	2
0xC005	SUB A,$1000	1	3
0xC008	INC X	1	1
0xC009	LD XL, A	1	1
0xC00A	SRL A	1	1
0xC00B	SWAP A	1	1
0xC00C	SLA $15	1	2
0xC00E	CP A,#$FE	1	2
0xC010	MOV $8000, #11	1	4
0xC014	MOV $8001, #22	1	4



F	Fetch	D	Decode	E	Execute

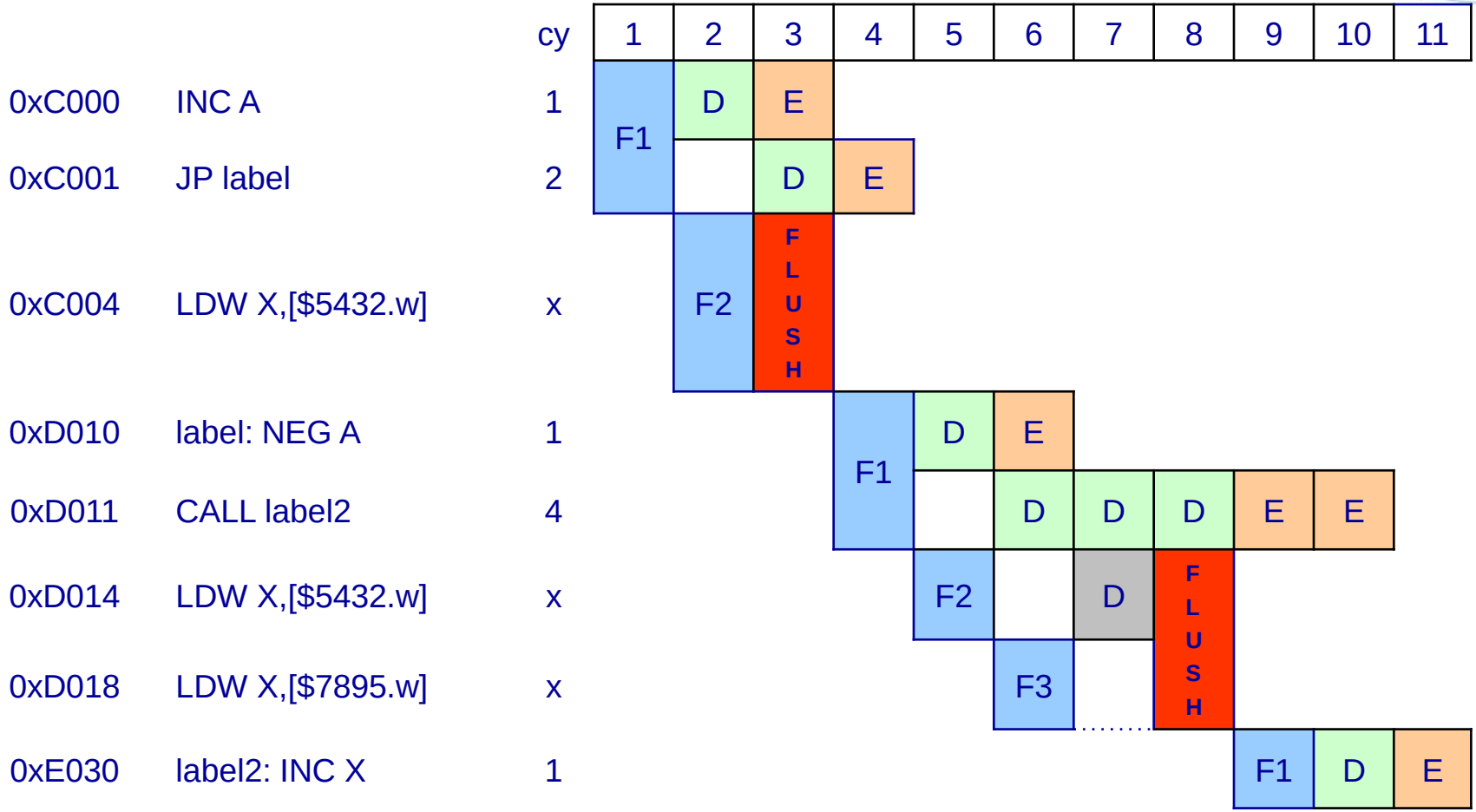
- Single cycle execution i.e. N instructions takes N clock cycles (CPI = 1)

Pipeline stalled example



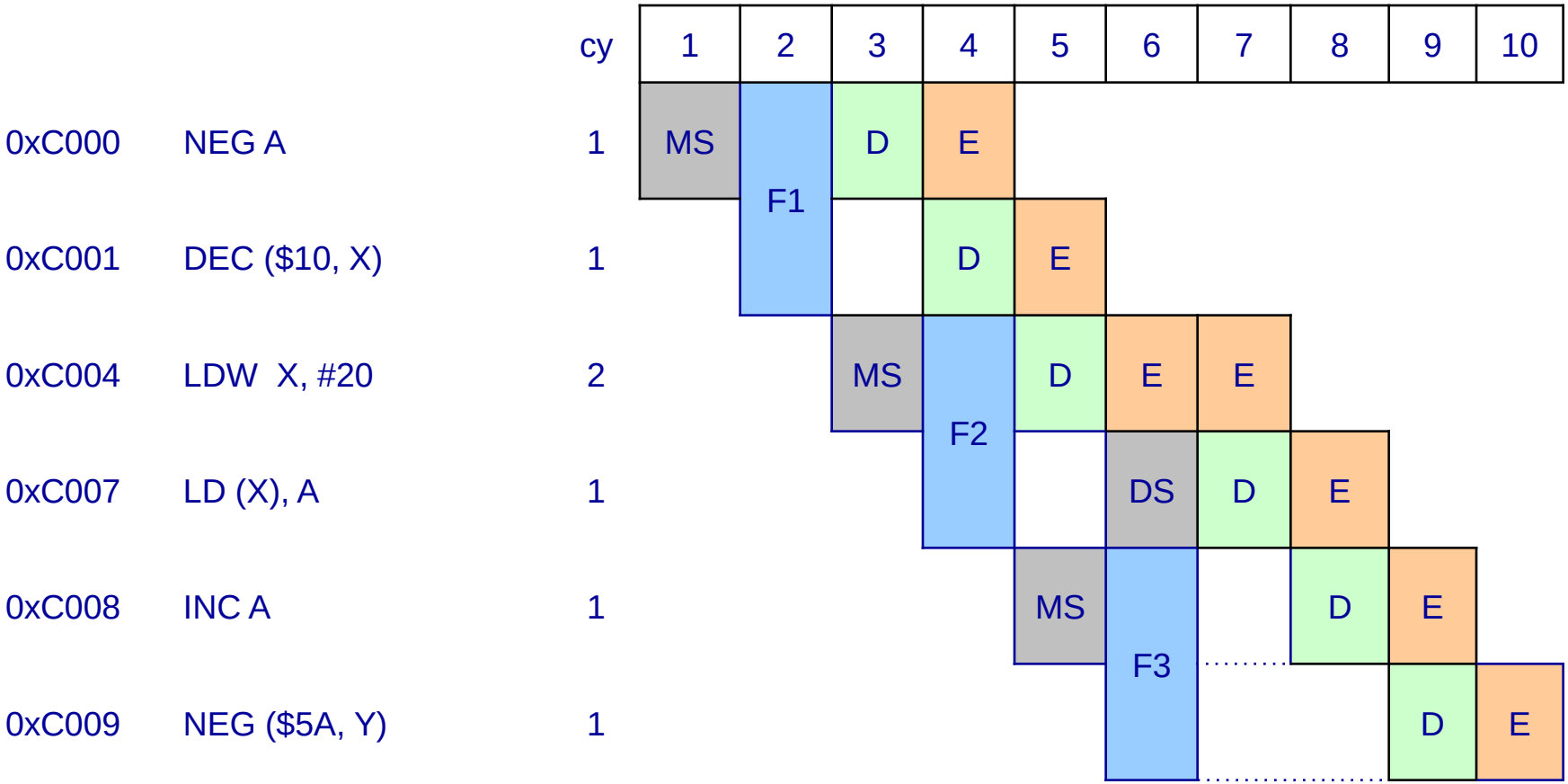
F	Fetch	D	Decode	E	Execute	D	Decode Stall

Pipeline with Call/Jump



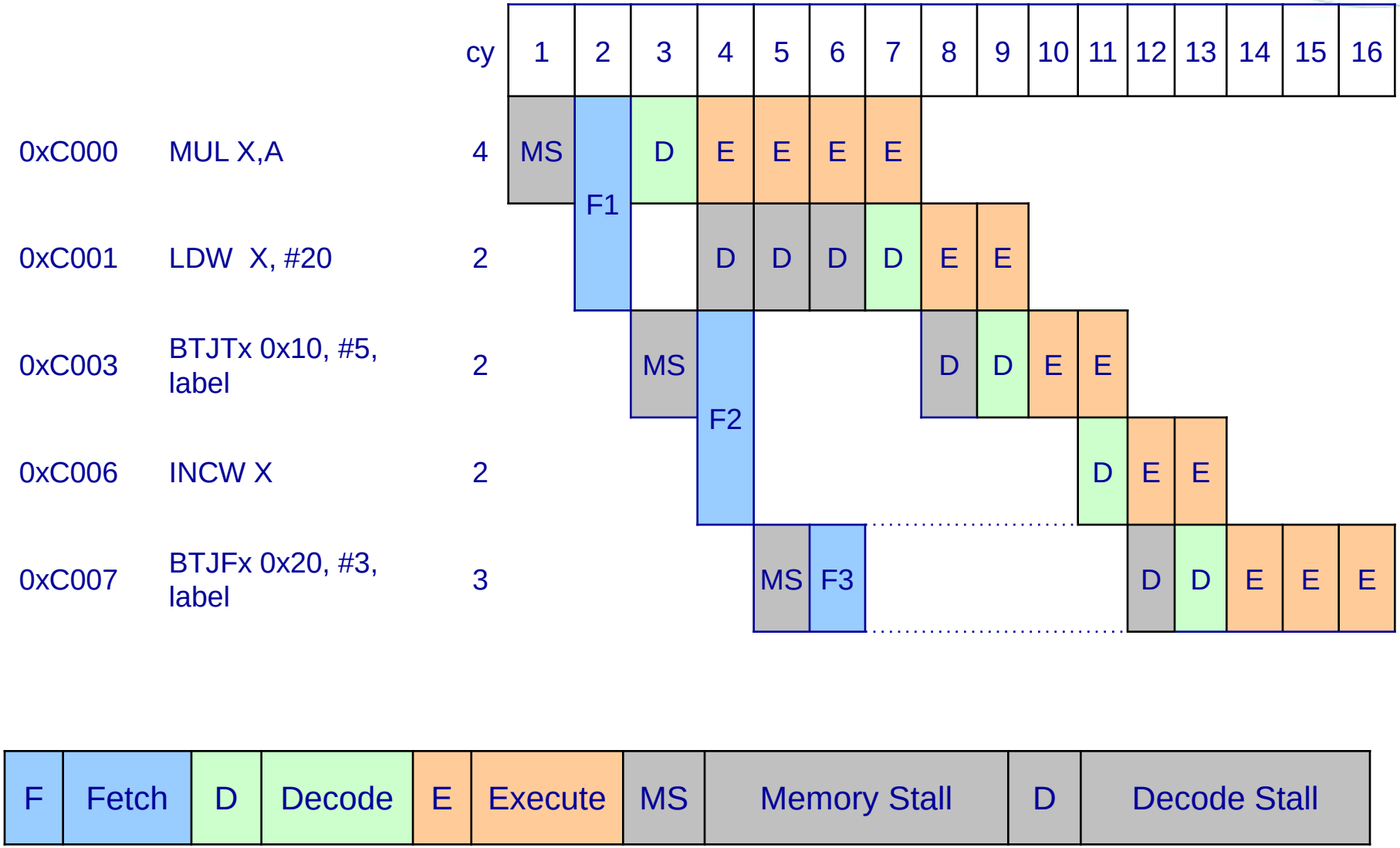
F	Fetch	D	Decode	E	Execute	D	Decode Stall

Optimal pipeline example with 1WS



F	Fetch	D	Decode	E	Execute	MS	Memory Stall

Pipeline stalled example with 1WS

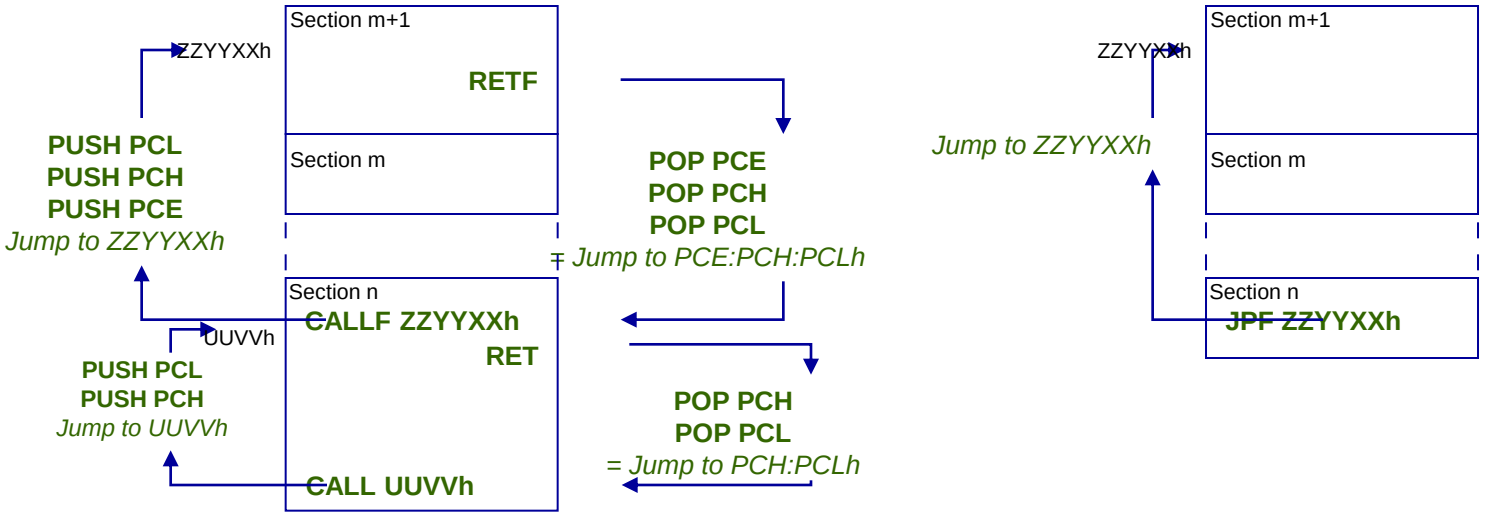


F	Fetch	D	Decode	E	Execute	MS	Memory Stall	D	Decode Stall

16MByte Memory Space Enhancements (1/

- **PROGRAM**: 16MByte linear memory space
  - Made up of 256 sections of 64-kbytes (example: section 0 is from 000000h to 00FFFFh)
  - 1 new CPU PCE register = Extended Program Counter

Program Counter is on 24-bit



New instructions to support this 24-bit PC

Instruction set

- **New addressing modes**
  - To take benefits of the 16-bit index registers
  - To access more efficiently the first 64k section
- **New instructions**
  - To handle 16-bit index registers
  - To improve compiler performance
  - To address the 16Mbyte memory space
  - To speed-up computation
    - Faster multiplication 8x8
    - 16-bit arithmetic instructions
  - To compute 16-bit operations
    - Division 16/8 and 16/16
  - To perform signed operation
    - V flag in CC

Addressing Modes (1/4)

MODE			SYNTAX	DEST. ADDRESS	POINTER ADDRESS	PTR. SIZE/ extra cycles
Inherent			NOP			
Immediate			LD A, #$55			
Short	Direct		LD A, label	0..255		
Long	Direct		LD A, label.w	0..64k		
Extended	**Direct**		LDF A, label	**0..16M**		
No offset	Direct	Indexed	LD A, (X)	0..64k		
Short	Direct	Indexed	LD A, ($10,X)	0..64k		
Short	Direct	SP indexed	LD A, ($10,SP)	0..255 + SP		
Long	Direct	Indexed	LD A, ($1000,X)	0..64k		
Extended	**Direct**	**Indexed**	LDF A, ($10000,X)	**0..16M**		

Addressing Modes (2/4)

MODE			SYNTAX	DEST. ADDRESS	POINTER ADDRESS	PTR. SIZE/ extra cycles
Short Pointer Short	Indirect		LD A, [$10]	0..255	0..255	1
Short Pointer Long	Indirect		LD A, [$10.w]	0..64k	0..255	2
Long Pointer Long	Indirect		LD A, [$1000.w]	0..64k	0..64k	2
Long Pointer Extended	**Indirect**		CALLF [$1000.w].e	**0..16M**	0..64k	3
Short Pointer Short	Indirect	Indexed	LD A,([$10],X)	0..64k	0..255	1
Short Pointer Long	Indirect	Indexed	LD A,([$10.w],X)	0..64k	0..255	2
Long Pointer Long	Indirect	Indexed (X only)	LD A,([$400.w],X)	0..64k	0..64k	2

Addressing Modes (3/4)

MODE			SYNTAX	DEST. ADDRESS	POINTER ADDRESS	PTR. SIZE/ extra cycles
Long Pointer Extended	**Indirect**	**Indexed**	LD A, ([$1000.e],X)	**0..16M**	**0..64k**	3
Relative	Direct (short)		JRNE loop	PC +/- 127		
Bit	Long Direct		BSET $1000, #7	0..64k		
Bit	Long Direct	Relative	BTJT $400,#7,skip	0..64k PC +/- 127		

Instruction set evolution - Brand new instructions

- **MOV**
  - To allow fast (1-cycle) copy from a memory location to another without affecting the accumulator
  - Supports immediate and direct addressing modes on short and long addresses

- **WFE**
  - To allow synchronization with other computing resources
  - CPU is stopped but internal peripherals still running
  - Execution will continue when an external event will occur
  - Internal interrupts are served according to I[0:1]

Instruction set evolution Related to index registers

	Instruction	Description	Cycles (w/o Fetch/Decode)
	LD X, Y / LDW X,Y	$X_{16} \leftarrow Y_{16}$	1
	LD S, Y / LDW S,Y	$SP \leftarrow Y_{16}$	1
	LD X, A / LD XL,A	$X_L \leftarrow A$, (X_H unaffected)	1
New	LD XH, A	$X_H \leftarrow A$, (X_l unaffected)	1
	LD X:A, SP	Removed (replaced by LDW X, SP)	-
	EXG X, Y / EXGW X,Y	$X_{16} \leftrightarrow Y_{16}$	1
	EXG A, Y	$A \leftrightarrow Y_L$, (Y_H unaffected)	1

- Instructions between 16 bits registers
- New instructions allow to handle one byte of the index registers

Instruction set
Related to index registers

Instruction	Description	Cycles
PUSH Y / PUSHW Y	$M(SP\text{--}) \leftarrow Y_L$ $M(SP\text{--}) \leftarrow Y_H$	2
SLL Y / SLLW Y	$Y_{16} \leftarrow Y_{16} << 1$ $CC.C \leftarrow Y.b_{15}$ $Y.b0 \leftarrow 0$	1
SWAP X /SWAPW X	$X_L \leftrightarrow X_H$	1
SWAP Y /SWAPW Y	$Y_L \leftrightarrow Y_H$	1
RLWA X,A (Y,A)	$A \leftarrow X_H \leftarrow X_L \leftarrow A$	1
RRWA X,A (Y,A)	$A \leftarrow X_L \leftarrow X_H \leftarrow A$	1

New

- Differences with index registers
  - Bytes are swapped instead of nibbles
- New RLWA and RRWA instructions
  - Very efficient to perform 8-bit data permutation

Instruction set evolution
16 bit Arithmetic

- 16-bit arithmetic added to X register (for address calculation)

Instruction	Description	Cycles
ADDW X, #imm16	X ← X + imm16	2
ADDW X, mem.w	X ← X + M(mem.w)*256 + M(mem.w+1)	2
ADDW X, (off.b,SP)	X ← X + M(SP+off.b)*256 + M(SP+off.b+1)	2
SUBW X, #imm16	X ← X - imm16	2
SUBW X, mem.w	X ← X - M(mem.w)*256 + M(mem.w+1)	2
SUBW X, (off.b,SP)	X ← X - M(SP+off.b)*256 + M(SP+off.b+1)	2

Instruction set evolution
Arithmetic operations

Instruction	Description	Cycles
LD X, [long.w] **LDW X, [long.w]**	Addr ← 256*M(long.w) + M(long.w + 1) X_H ← M(Addr) X_L ← M(Addr + 1)	2
CP X, (shortoff,SP) **CPW X, (shortoff,SP)**	Test {X – M(SP+short):M(SP+short+1)Addr);	2
MUL Y, A / FMUL Y,A	Y ← Y_L * A	4
DIV X,A	X ← X / A, A ← X % A	16
DIVW X,Y	X ← X / Y, Y ← X % Y	16

- Fast handling of integers
- Improved multiplying and dividing
  - With fast multiplying 8 by 8
  - Dividing 16 by 8 and 16 by 16

Instruction set evolution
Function parameter passing

Instruction	Description	Cycles
NEG (off.b,SP)	M(SP+off.b) ← not (M(SP+off.b)) + 1	2
CPL (off.b,SP)	M(SP+off.b) ← not M(SP+off.b)	2
SRL (off.b,SP)	M(SP+off.b) ← M(SP+off.b) >> 1	2
RRC (off.b,SP)	M(SP+off.b) ← CC.C*2^8 + (M(SP+off.b) >> 1)	2
SRA (off.b,SP)	M(SP+off.b) ← (M(SP+off.b) .b7*2^8 + (M(SP+off.b) >> 1)	2
SLL (off.b,SP)	M(SP+off.b) ← M(SP+off.b) <<1	2
RLC (off.b,SP)	M(SP+off.b) ← M(SP+off.b) <<1 + CC.C	2
DEC (off.b,SP)	M(SP+off.b) ← M(SP+off.b) - 1	2
INC (off.b,SP)	M(SP+off.b) ← M(SP+off.b) + 1	2
TNZ (off.b,SP)	Test M(SP+off.b)	1
SWAP (off.b,SP)	M(SP+off.b) ← M(SP+off.b) <<4 + M(SP+off.b) <<4	2
CLR (off.b,SP)	M(SP+off.b) ← 0	1

- All these instructions allow the compiler to generate a more efficient code

Instruction set evolution
Stack pointer operations

Instruction	Description	Cycles
ADD SP, #n	SP ← SP + n	1
SUB SP,#n	SP ← SP - n	1

- Immediate addition/substraction to SP
  - Allow fast allocation/de-allocation of parameters on stack
- These 2 instructions allow the compiler to generate a more efficient code

Instruction set evolution
Signed Arithmetic

- Added V in CC register flag for signed arithmetic
  - JRSGT, JRSLE, JRSGE, JRSLT, JRNV, JRV
  - RVF

Instruction	Description	Cycles
JRSGT	Jump relative if signed greater than Condition: Z or (N xor V)=0	1/2
JRSLE	Jump relative if signed greater than Condition: Z or (N xor V)=1	1/2
JRSGE	Jump relative if signed greater than Condition: (N xor V)=0	1/2
JRSLT	Jump relative if signed greater than Condition: (N xor V)=1	1/2
JRNV	Jump relative if not overflow Condition: V = 0	1/2
JRV	Jump relative if overflow Condition: V = 1	1/2
RVF	CC.V ← 0	1

Performance

- **Most instructions are executed in a single cycle**
  - Vs 3 to 4 cycles/instructions for ST7
- **Improved code efficiency (~30% smaller vs ST7)**
- **Leading edge 8-bit controller**
  - 0.29 DMIPS – similar to some well known 16-bit CPU performance
  - Twice as fast as 8-bit most used CPU (following standard industry benchmark)

STM8 vs ST7

Features	ST7	STM8
Addr Range	64KB Linear	**16MB Linear**
Architecture	Von Neumann CISC	**Harvard** **CISC**
Clocks per Instr. (CPI)	3 to 12	**3 Stage Pipe Line** **Avg 1.6 CPI**
ALU	8-bit	**8-bit &** **16-bit for address calculation**
Stack	8-bit SP	**16-bit SP with addr modes & stack/param allocation**
Other	8x8 Mul Bit handling on first page	**16-bit index pointers** **Long relative branch** **Fast 8x8 Mul;** **16/8 & 16/16 Division;** **Bit Handling on first section;** **Adv Debug Support**
Low Power	None	**Reduced data/fetch transfers; High instruction efficiency**

References

PM0044

STM8 CPU Programming Manual

QUIZ

- What is new in core architecture ?
- What is the length of the index registers ?
- How many CPU cycles are needed for most of operations ?

New debug capabilities

- **SWIM** : **S**ingle **W**ire **I**nterface **M**odule
  - SWIM is both a protocol and a a communication cell with bus access capability
  - Faster protocol, running at HSI frequency
  - Cell can read on the fly without stalling the CPU
    - No more monitor using CPU resources
  - New shadow core registers
    - could be readed by SWIM on the fly without core stall
  - A pin is dedicated to SWIM communication
    - Can also be used as GPIO
    - Configuration in the CFG Global Configuration Register (CFG_GCR)

SWIM vs ICC

Functions	ICC	SWIM
Connections	VDD, VSS, Reset, IccData, IccClk, (VPP), (ext clock)	**VDD, VSS, Reset, SwimDbg**
Baud rate	20 kbytes at 8MHz	**144 kbytes at 16MHz**
Read/Write on the fly	intrusive	**RAM, Peripherals & (shadow) Core registers non intrusive**
Software monitor/TRAP interrupt reserved	Always	**Not needed**
Software breakpoints	TRAP	**SWBRK**
Hot plug (debug without reset)	Not possible	**Yes**, in specification phase

Non-intrusive SWIM access

Non-intrusive

- SWIM doesn't use any CPU resource. No restrictions for addresses and memory space.
  - No monitor code
  - No interrupt remapping

Real-time code execution

- SWIM steals dead cycles to read RAM and registers.
- Write accesses need to stop CPU execution.

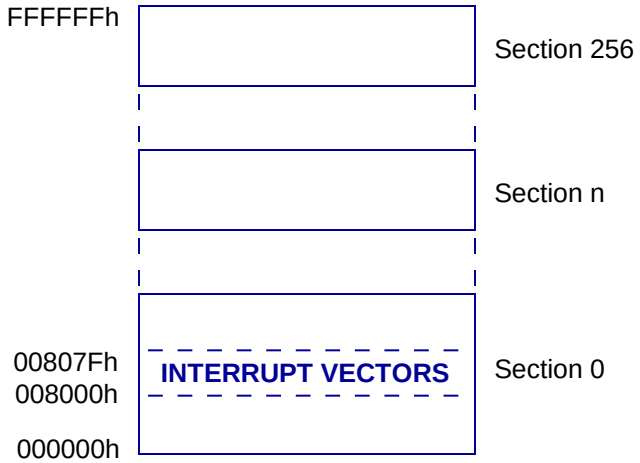
Interrupt management (1/4)

- **Up to 32 interrupt vectors**
  - At least one interrupt by peripheral
- **Nested or concurrent modes**
  - Software priorities programmable by ISPR registers
    - 3 levels + disable interrupts
- **A Top Level Interrupt (TLI)**
  - Managed by PD7
  - 2 sensitivities : falling edge or rising edge
  - Enabled by software
  - TLI can only be interrupted by TRAP or RESET
- **5 external interrupts**
  - linked to a port from PA to PE
    - No status register, no flag to clear
  - Programmable sensitivity
- **Boot mode support**
  - For memory update (by UART, CAN, LIN)

Interrupt Management (2/4)

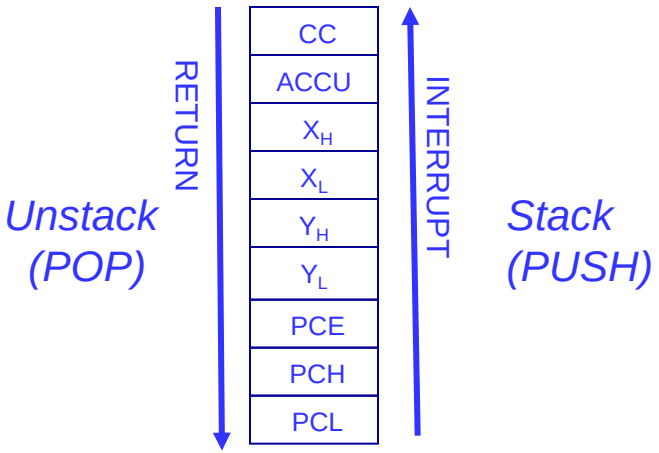
- **Interrupt vector table**
  - 32 extended vectors
  - Address >> 08000h-0807Fh on STM8S family
  - 3-byte interrupt vector = interrupt routine anywhere in the memory space

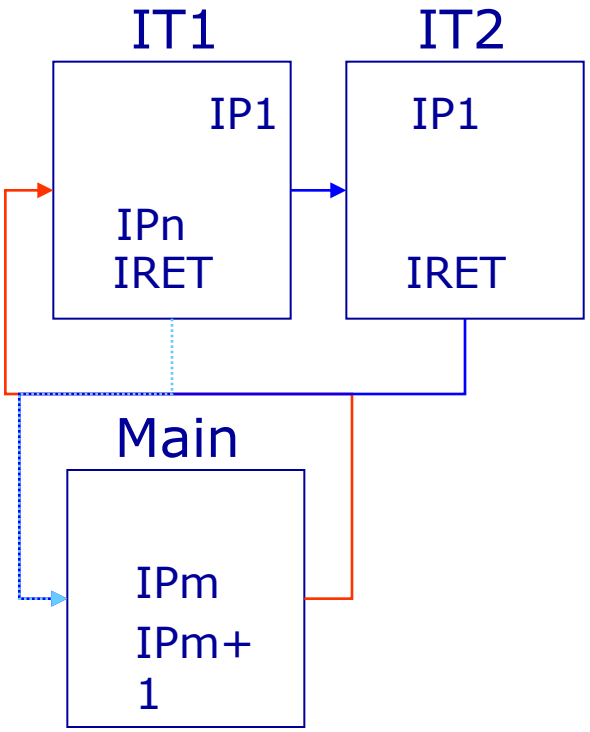
    Note: each vector table entry has 4-bytes – 1st byte contains the dedicated instruction opcode (82h)
  - Breakpoint not supported in the vector table



Interrupt Management (3/4)

- **Interrupt and stack management**
  - full context save (A,X,Y,CC, PC)
  - 4 nested interrupt levels (with TLI and TRAP)
  - Tail chaining





Interrupt Management (4/4)

- **Interrupt latency**
  - 9 cycles needed to save the context
  - Instruction in Exectue stage is always completed
    - Except for divisions (DIV and DIVW) which can be interrupted.
  - Some instructions shares Decode/Execution stage
    - Those cannot be interrupted even in decode
    - Typically instructions (re)storing context or accessing 16bit X/Y
  - Max latency is 15 cycles in case of CALLF indirect execution ($8T_{cpu}$) (context saving start on the last but one execution cycle)

- **Interrupt and fetch**
  - The current fetch and decode pipeline contents are lost after return from interrupt.

Quiz

- Where are located the interrupt vectors and how many are they ?
- What is the most prior external interrupt and what are the differences versus standard external interrupts ?
- What is stacked when an interrupt is served ?

CONFIDENTIALITY OBLIGATIONS:

THIS DOCUMENT CONTAINS SENSITIVE INFORMATION.

IT IS CLASSIFIED “MICROCONTROLLERS, MEMORIES & SECURE MCUs (MMS) RESTRICTED AND ITS DISTRIBUTION IS SUBMITTED TO ST/MMS AUTHORIZATION

AT ALL TIMES YOU SHOULD COMPLY WITH THE FOLLOWING SECURITY RULES:

- DO NOT COPY OR REPRODUCE ALL OR PART OF THIS DOCUMENT
- KEEP THIS DOCUMENT LOCKED AWAY
- FURTHER COPIES CAN BE PROVIDED ON A “NEED TO KNOW BASIS”, PLEASE CONTACT YOUR LOCAL ST SALES OFFICE OR DOCUMENT WRITTER.