Lecture 2
MSP430 Architecture

Texas Instruments Incorporated
University of Beira Interior (PT)

Pedro Dinis Gaspar, António Espírito Santo, Bruno Ribeiro, Humberto Santos
University of Beira Interior, Electromechanical Engineering Department
www.msp430.ubi.pt

>> Contents

Contents (1/2)

Contents (2/2)

Introduction

- **A comprehensive description of the MSP430 architecture, covering its:**
  - Main characteristics;
  - Device architecture;
  - Address space;
  - Interrupt vector table;
  - Central Processing Unit (MSP430 CPU and MSP430X CPU);
  - 7 seven addressing modes and instruction set composed of:
    - 27 base opcodes;
    - 24 emulated instructions.

Microcontroller characteristics

- **Integration:** Able to implement a whole design onto a single chip.
- **Cost:** Are usually low-cost devices (a few $ each);
- **Clock frequency:** Compared with other devices (microprocessors and DSPs), MCUs use a low clock frequency:
  - MCUs today run up to 100 MHz/100 MIPS (Million Instructions Per Second).
- **Power consumption:** Low power (battery operation);
- **Bits:** 4 bits (older devices) to 32 bits devices;
- **Memory:** Limited available memory, usually less than 1 MByte;
- **Input/Output (I/O):** Low to high (8 to 150) pin-out count.

MSP430 main characteristics (1/3)

- **Low power consumption:**
  - 0.1 μA for RAM data retention;
  - 0.8 μA for real-time clock mode operation;
  - 250 μA/MIPS during active operation.
- **Low operation voltage (from 1.8 V to 3.6 V);**
- **< 1 μs clock start-up;**
- **< 50 nA port leakage;**
- **Zero-power Brown-Out Reset (BOR).**

MSP430 main characteristics (2/3)

- **On-chip analogue features:**
  - 10/12/16-bit Analogue-to-Digital Converter (ADC);
  - 12-bit dual Digital-to-Analogue Converter (DAC);
  - Comparator-gated timers;
  - Operational Amplifiers (Op Amps);
  - Supply Voltage Supervisor (SVS).

- **16 bit RISC CPU:**
  - Compact core design reduces power consumption and cost;
  - 16-bit data bus;
  - 27 core instructions;
  - 7 addressing modes;
  - Extensive vectored-interrupt capability.

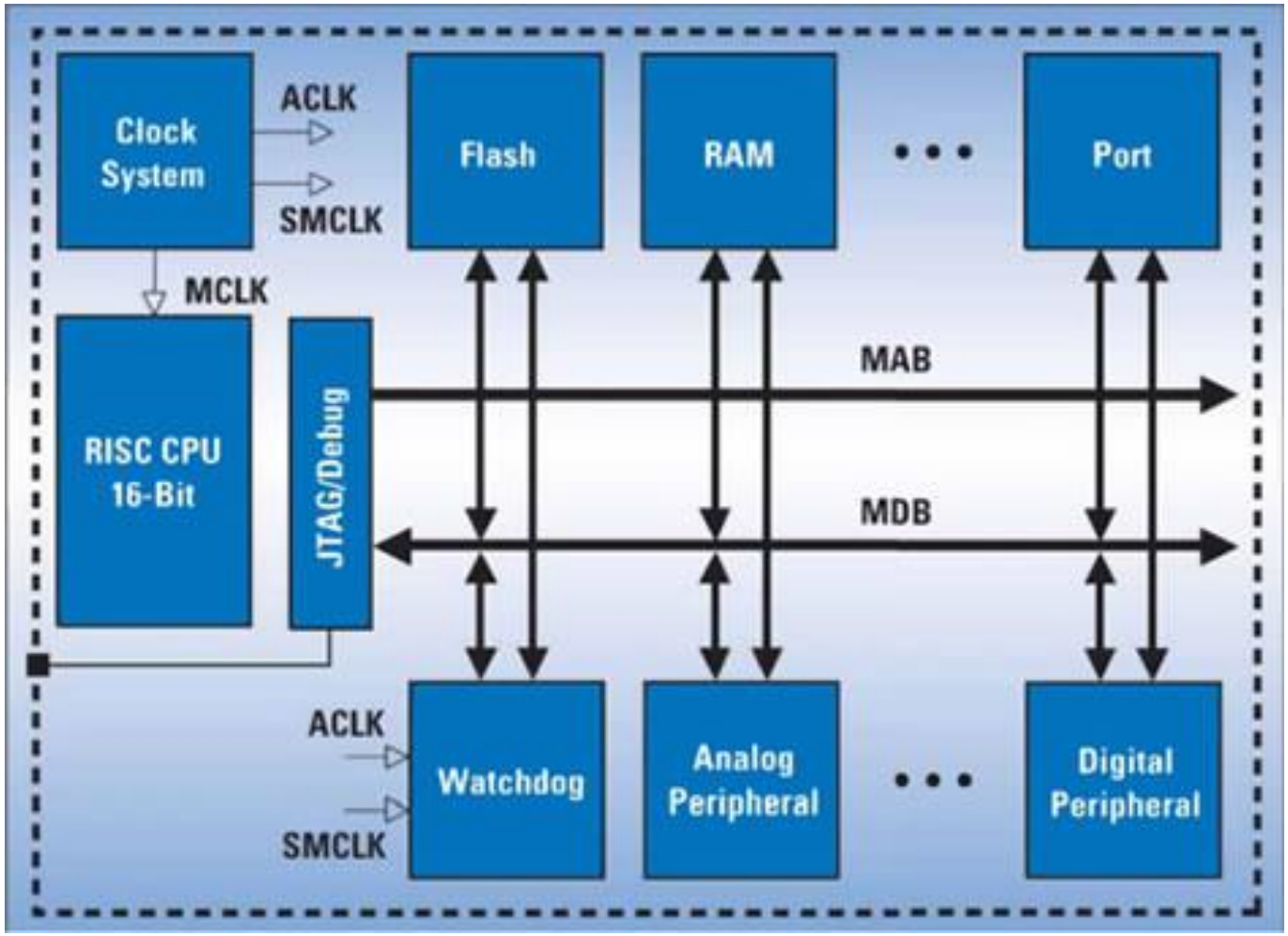
MSP430 main characteristics (3/3)

- **Flexibility:**
  - Up to 256 kByte Flash;
  - Up to 100 pins;
  - USART, I2C, Timers;
  - LCD driver;
  - Embedded emulation;
  - And many more peripherals modules…

- **Microcontroller performance:**
  - Instruction processing on either bits, bytes or words
  - Reduced instructions set;
  - Compiler efficient;
  - Wide range of peripherals;
  - Flexible clock system.

MSP430 Architecture

- **Block diagram:**



Address Space

- **Mapped into a single, contiguous address space:**
  - **Flash/ROM:** All code, tables, and hard-coded constants reside in this memory space.
  - **Information memory:** Variables needed for the next power up can be stored here during power down. It can also be used as code memory.
  - **Boot memory:** The bootstrap loader performs some of the same functions as the JTAG interface.
  - **RAM:** RAM is used for both code and data.
  - **Peripheral Modules:** Peripheral modules consist of all on-chip peripheral registers that are mapped into the address space.
  - **Special Function Registers (SFRs):** Some peripheral functions are mapped into memory with special dedicated functions.

Memory Map

Memory Address		Description	Access
End:	0FFFFh	**Interrupt Vector Table**	Word/Byte
Start:	0FFE0h		
End:	0FFDFh	**Flash/ROM**	Word/Byte
Start *:	0F800h		
	01100h		
End *:	010FFh	**Information Memory (Flash devices only)**	Word/Byte
	0107Fh		
Start:	01000h		
End:	0FFFh	**Boot Memory (Flash devices only)**	Word/Byte
Start:	0C00h		
End *:	09FFh	**RAM**	Word/Byte
	027Fh		
Start:	0200h		
End:	01FFh	**16-bit Peripheral modules**	Word
Start:	0100h		
End:	00FFh	**8-bit Peripheral modules**	Byte
Start:	0010h		
End:	000Fh	**Special Function Registers**	Byte
Start:	0000h		

Interrupt vector table

- **Mapped at the very end of memory space (upper 16 words of Flash/ROM):** `0FFE0h – 0FFFEh` (4xx devices);
- **Priority of the interrupt vector increases with the word address.**

Vector Address	Priority	'2xx	'4xx
0xFFFE	31, Highest	**Hard Reset/ Watchdog**	**Hard Reset/ Watchdog**
0xFFFC	30	**Oscillator/ Flash/NMI**	**Oscillator/ Flash/NMI**
0xFFFA	29	**Timer_B** (22x2, 22x4, 23x, 24x, 26x)	Timer_B ('43x and'44x)
0xFFF8	28	**Timer_B** (22x2, 22x4, 23x, 24x)	Timer_B ('43x and'44x)
0xFFF6	27	**Comparator_A+** (20x1, 21x1, 23x, 24x, 26x)	**Comparator**
0xFFF4	26	**Watchdog Timer+**	**Watchdog Timer**
0xFFF2	25	**Timer_A**	**USART0 Rx** ('43x and'44x)
0xFFF0	24	**Timer_A**	**USART0 Tx** ('43x and'44x)
0xFFEE	23	**USCI Rx** (22x2, 22x4, 23x, 24x, 26x) **I2C status** (23x, 24x)	**ADC** ('43x and'44x)
0xFFEC	22	**USCI Tx** (22x2, 22x4, 23x, 24x, 26x) **I2C Rx/Tx** (23x, 24x, 26x)	**Timer_A**
0xFFEA	21	**ADC10** (20x2 22x2, 22x4) **ADC12** (23x, 24x, 26x) **SD16_A** (20x3)	**Timer_A**
0xFFE8	20	**USI** (20x2, 20x3)	**Port 1**
0xFFE6	19	**Port P2**	**USART1 Rx** ('44x)
0xFFE4	18	**Port P1**	**USART1 Tx** ('44x)
0xFFE2	17	**USCI Rx** (23x, 24x, 26x) **I2C status** (241x, 261x)	**Port 2**
0xFFE0	16	**USCI Tx** (23x,24x) **I2C Rx/Tx** (241x, 261x)	**Basic Timer**
	15	**DMA** (241x, 261x)	
	14	**DAC12** (241x, 261)	
	13 to 0, Lowest	**Reserved**	

Central Processing Unit (MSP430 CPU) (1/7)

- **RISC (Reduced Instructions Set Computing) architecture:**
  - Instructions are reduced to the basic ones (short set):
    - 27 physical instructions;
    - 24 emulated instructions.
  - This provides simpler and faster instruction decoding;
  - Interconnect by a using a common memory address bus (MAB) and memory data bus (MDB) - Von Neumann architecture:
    - Makes use of only one storage structure for data and instructions sets.
    - The separation of the storage processing unit is implicit;
    - Instructions are treated as data (programmable).

Central Processing Unit (MSP430 CPU) (2/7)

- **RISC (Reduced Instructions Set Computing) type architecture:**
  - Uses a 3-stage instruction pipeline containing:
    - Instruction decoding;
    - 16 bit ALU;
    - 4 dedicated-use registers;
    - 12 working registers.

- **Address bus has 16 bit so it can address 64 kB (including RAM + Flash + Registers);**

- **Arithmetic Logic Unit (ALU):**
  - Addition, subtraction, comparison and logical (AND, OR, XOR) operations;
  - Operations can affect the overflow, zero, negative, and carry flags of the SR (Status Register).

Central Processing Unit (MSP430 CPU) (3/7)

- **Incorporates sixteen 16-bit registers:**
  - 4 registers (R0, R1, R2 and R3) have dedicated functions;
  - 12 register are working registers (R4 to R15) for general use.

- **R0: Program Counter (PC):**
  - Points to the next instruction to be read from memory and executed by the CPU.

- **R1: Stack Pointer (SP):**
  - 1st: stack can be used by user to store data for later use (instructions: store by PUSH, retrieve by POP);
  - 2nd: stack can be used by user or by compiler for subroutine parameters (PUSH, POP in calling routine; addressed via offset calculation on stack pointer (SP) in called subroutine);

Central Processing Unit (MSP430 CPU) (4/7)

- **R1: Stack Pointer (SP) (continued):**
  - 3rd: used by subroutine calls to store the program counter value for return at subroutine's end (RET);
  - 4th: used by interrupt - system stores the actual PC value first, then the actual status register content (on top of stack) on return from interrupt (RETI) the system get the same status as just before the interrupt happened (as long as none has changed the value on TOS) and the same program counter value from stack.

Central Processing Unit (MSP430 CPU) (5/7)

R2: Status Register (SR):

- Stores status and control bits;
- System flags are changed automatically by the CPU;
- Reserved bits are used to support the constant generator.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved for CG1							V	SCG1	SCG0	OSCOFF	CPUOFF	GIE	N	Z	C

Bit		Description
8	V	Overflow bit. V = 1 ⇒ Result of an arithmetic operation overflows the signed-variable range.
7	SCG1	System clock generator 0. SCG1 = 1 ⇒ DCO generator is turned off – if not used for MCLK or SMCLK
6	SCG0	System clock generator 1. SCG0 = 1 ⇒ FLL+ loop control is turned off
5	OSCOFF	Oscillator Off. OSCOFF = 1 ⇒ turns off LFXT1 when it is not used for MCLK or SMCLK
4	CPUOFF	CPU off. CPUOFF = 1 ⇒ disable CPU core.
3	GIE	General interrupt enable. GIE = 1 ⇒ enables maskable interrupts.
2	N	Negative flag. N = 1 ⇒ result of a byte or word operation is negative.
1	Z	Zero flag. Z = 1 ⇒ result of a byte or word operation is 0.
0	C	Carry flag. C = 1 ⇒ result of a byte or word operation produced a carry.

Central Processing Unit (MSP430 CPU) (6/7)

R2/R3: Constant Generator Registers (CG1/CG2):

- Depending of the source-register addressing modes (As) value, six constants can be generated without code word or code memory access to retrieve them.

- This is a very powerful feature which allows the implementation of emulated instructions, for example, instead of implement a core instruction for an increment the constant generator is used.

Register	As	Constant	Remarks
R2	00	-	Register mode
R2	01	(0)	Absolute mode
R2	10	00004h	+4, bit processing
R2	11	00008h	+8, bit processing
R3	00	00000h	0, word processing
R3	01	00001h	+1
R3	10	00002h	+2, bit processing
R3	11	0FFFFh	-1, word processing

Central Processing Unit (MSP430 CPU) (7/7)

- **R4 - R15: General–Purpose Registers:**
  - These general-purpose registers are adequate to store data registers, address pointers, or index values and can be accessed with byte or word instructions.

Addressing modes

- **7 addressing modes for the source operand:**

Register Mode	`mov.w R10,R11` Single cycle
Indexed Mode	`mov.w 2(R5),6(R6)` Table processing
Symbolic Mode	`mov.w EDE,TONI` Easy to read code, PC relative
Absolute Mode	`mov.w &EDE,&TONI` Directly access any memory location
Indirect Register Mode	`mov.w @R10,0(R11)` Access memory with pointers
Indirect Autoincrement	`mov.w @R10+,0(R11)` Table processing
Immediate Mode	`mov.w #45h,&TONI` Unrestricted constant values

- **4 addressing modes for the destination operand:**
  - Register mode; Indexed mode; Symbolic mode; Absolute mode.

- **For the destination operand, two additional addressing modes can be emulated.**

Instruction set

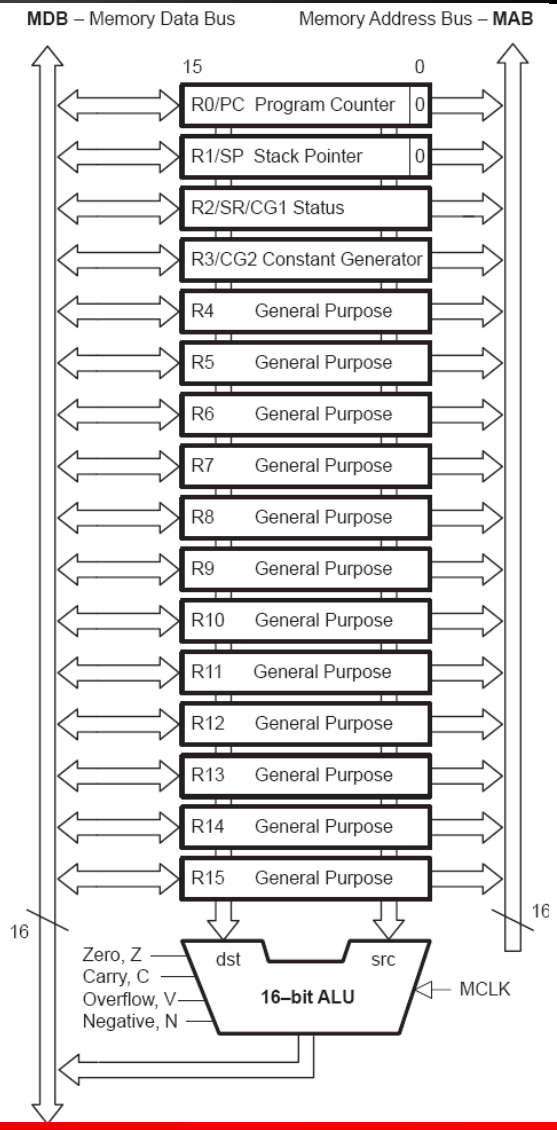
- **27 core instructions;**
- **24 emulated instructions;**
- **The instruction set is orthogonal;**
- **The core instructions have unique opcodes decoded by the CPU, while the emulated ones need assemblers and compilers for their mnemonics;**
- **There are three core-instruction formats:**
  - Double operand;
  - Single operand;
  - Program flow control - Jump.

Exploring the addressing modes of the MSP430 architecture (1/11)

- **The MSP430 architecture:**
  - The MSP430 CPU incorporates features specifically designed to allow the use of modern programming techniques, such as:
    - The computation of jump addresses;
    - Data processing in tables;
    - The use of high-level languages such as C.
  - The whole memory space can be addressed by the MSP430 CPU, without needing paging, using seven different addressing modes.
  - The MSP430 CPU has a set of 27 instructions that can be used with any of the addressing modes.

Exploring the addressing modes of the MSP430 architecture (2/11)

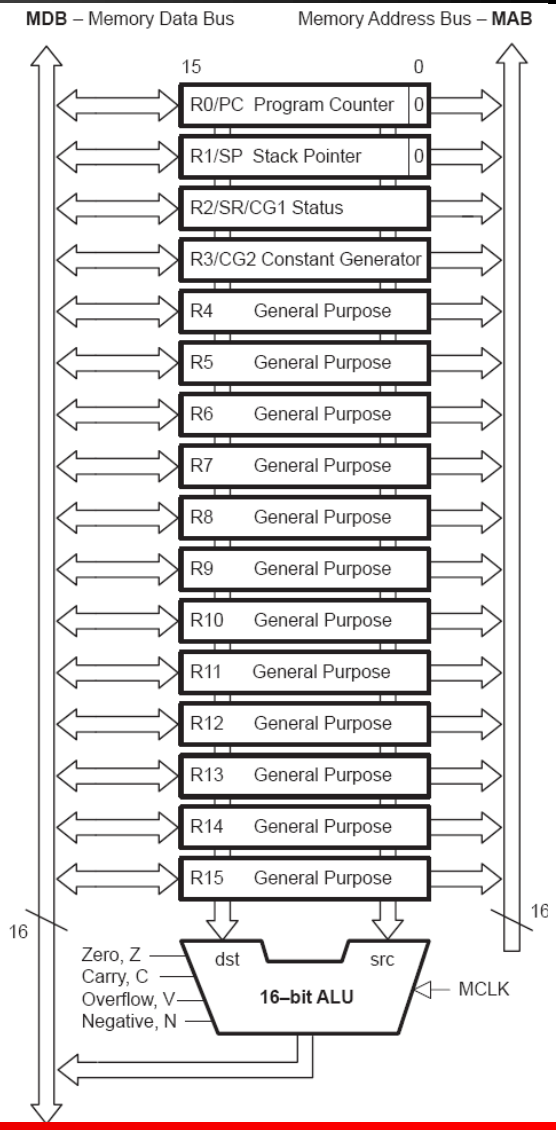
- **Organization of the MSP430 CPU:**
  - 16-bit address bus (MAB) and 16-bit data bus (MDB).
  - Both registers and memory can be accessed either in word format or in byte format.
  - Allows the direct transfer of data between memory, without passing through the registers.
  - The 16-bit registers can be accessed directly through the instructions, some of which run in a single clock cycle.
  - Some of the constants most used in programming can be obtained from the constant generator.



Exploring the addressing modes of the MSP430 architecture (3/11)

Organization of the MSP430 CPU (continued):

- The architecture has a 16 bit Arithmetic Logic Unit (ALU).
- Carrying out operations affects the state of the following flags:
  - Zero (Z);
  - Carry (C);
  - Overflow (V);
  - Negative (N).
- The MCLK (Master) clock signal drives the CPU.



Exploring the addressing modes of the MSP430 architecture (4/11)

- **The MSP430 CPU has 16 registers, some of which are dedicated to special use:**

- **R0 (PC) (Program Counter):**
  - This register always points to the next instruction to be fetched;
  - Each instruction occupies an even number of bytes. Therefore, the least significant bit (LSB) of the PC register is always zero;
  - After fetch of an instruction, the PC register is incremented to point to the next instruction.

Exploring the addressing modes of the MSP430 architecture (5/11)

- **R1 (SP) Stack Pointer:**
  - This register is used by the MSP430 CPU to store the return address of routines or interrupts;
  - Each time the data stack is accessed, the SP is incremented or decremented automatically;
  - The user should be careful to initialise the SP register with the valid address of the data stack in RAM;
  - The stack pointer SP always points to an even address, so its LSB is always zero.

Exploring the addressing modes of the MSP430 architecture (6/11)

❑ R2 – SR/CG1 & R3 – CG2 (Status Register):

- The status of the MSP430 CPU is defined by a set of bits contained in register R2 (SR) (status register);
- This register can only be accessed through register addressing mode;
- All other addressing modes are reserved to support the constants generator;
- The organization of the individual bits of register R2 is shown in the following figure:

Exploring the addressing modes of the MSP430 architecture (7/11)

❑ R2 – SR/CG1 & R3 – CG2 (Status Register):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved for CG1							V	SCG1	SCG0	OSCOFF	CPUOFF	GIE	N	Z	C

Bit		Description
8	V	Overflow bit. V = 1 ⇒ Result of an arithmetic operation overflows the signed-variable range.
7	SCG1	System clock generator 1. SCG1 = 1 ⇒ DCO generator is turned off – if not used for MCLK or SMCLK
6	SCG0	System clock generator 0. SCG0 = 1 ⇒ FLL+ loop control is turned off
5	OSCOFF	Oscillator Off. OSCOFF = 1 ⇒ turns off LFXT1 when it is not used for MCLK or SMCLK
4	CPUOFF	CPU off. CPUOFF = 1 ⇒ disable CPU core.
3	GIE	General Interrupt Enable. GIE = 1 ⇒ enables maskable interrupts.
2	N	Negative flag. N = 1 ⇒ result of a byte or word operation is negative.
1	Z	Zero flag. Z = 1 ⇒ result of a byte or word operation is 0.
0	C	Carry flag. C = 1 ⇒ result of a byte or word operation produced a carry.

Exploring the addressing modes of the MSP430 architecture (8/11)

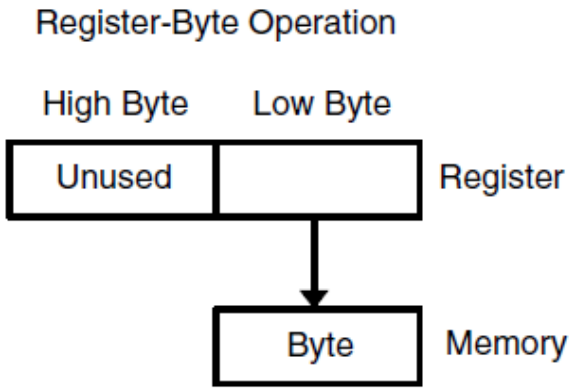
R2 – SR/CG1 & R3 – CG2 (Constant Generators):

- Six different constants commonly used in programming can be generated using the registers R2 and R3, without the need to add a 16-bit word of code to the instruction;
- The constants are fixed and are selected by the instruction bits (As). These control the addressing mode.

Register	As	Constant	Remarks
R2	00	-	Register mode
R2	01	(0)	Absolute mode
R2	10	00004h	+4, bit processing
R2	11	00008h	+8, bit processing
R3	00	00000h	0, word processing
R3	01	00001h	+1
R3	10	00002h	+2, bit processing
R3	11	0FFFFh	-1, word processing

Exploring the addressing modes of the MSP430 architecture (9/11)

- **R4-R15: General-purpose registers:**
  - The general purpose registers R4 to R15 can be used as data registers, data pointers and indices. They can be accessed either as a byte or as a word;
  - These registers support operations on words or bytes;
  - In the example to the right, the contents of the least significant byte of register R5 (0x8F) is added to the contents of the memory address pointed to by the register R6 (0x12h);



Example Register-Byte Operation

R5 = 0A28Fh

R6 = 0203h

Mem(0203h) = 012h

```
ADD.B     R5,0(R6)
```

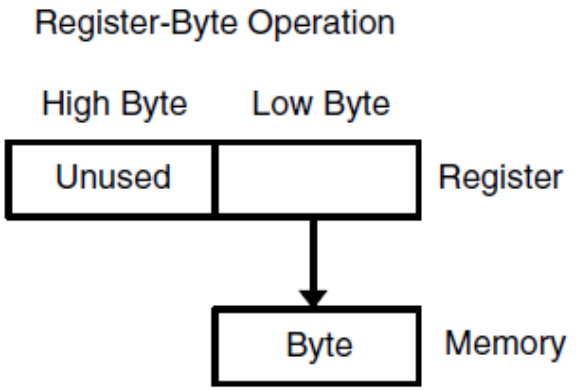
```
   08Fh
 + 012h
 -------
   0A1h
```

Mem (0203h) = 0A1h

C = 0, Z = 0, N = 1

Exploring the addressing modes of the MSP430 architecture (10/11)

- **R4-R15: General-purpose registers:**
  - The contents of the memory address is updated with the result of the operation (0xA1);
  - The status flags of the CPU in the SR are updated after the execution of the instruction.



Example Register-Byte Operation

R5 = 0A28Fh

R6 = 0203h

Mem(0203h) = 012h

```
ADD.B     R5,0(R6)
```

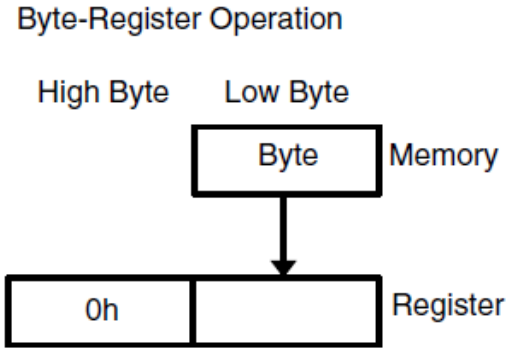
```
  08Fh
+ 012h
------
  0A1h
```

Mem (0203h) = 0A1h

C = 0, Z = 0, N = 1

Exploring the addressing modes of the MSP430 architecture (11/11)

- **R4-R15: General-purpose registers:**
  - The contents of the memory address pointed to by R6 (0x5F) is added to the contents of the least significant byte of register R5 (0x02);
  - The result of the operation (0x61) is stored in the least significant byte of register R5;
  - Meanwhile, the most significant byte of register R5 is set to zero;
  - The system flags in SR are updated in accordance with the result.



Example Byte-Register Operation

R5 = 01202h

R6 = 0223h

Mem(0223h) = 05Fh

```
ADD.B       @R6,R5
```

```
    05Fh
  + 002h
  -------
  00061h
```

R5 = 00061h

C = 0, Z = 0, N = 0

Instructions format (1/4)

- **In addition to the 27 instructions of the CPU there are 24 emulated instructions;**
- **The CPU coding is unique;**
- **The emulated instructions make reading and writing code more easy, but do not have their own op-codes;**
- **Emulated instructions are replaced automatically by instructions from the CPU;**
- **There are no penalties for using emulated instructions.**
- **There are three formats used to encode instructions for processing by the CPU core:**
  - Double operand;
  - Single operand;
  - Jumps.

Instructions format (2/4)

- The instructions for double and single operands, depend on the suffix used, (`.W`) word or (`.B`) byte.
- These suffixes allow word or byte data access;
- If the suffix is ignored, the instruction processes word data by default.

Instructions format (3/4)

- **The source and destination of the data operated by an instruction are defined by the following fields:**
  - **src**: source operand address, as defined in As and S-reg;
  - **dst**: destination operand address, as defined in Ad and D-reg;
  - **As**: addressing bits used to define the addressing mode used by the source operand;
  - **S-reg**: register used by the source operand;
  - **Ad**: Addressing bits used to define the addressing mode used by the destination operand;
  - **D-reg**: register used by the destination operand;
  - **B/W**: word or byte access definition bit.

Instructions format (4/4)

- While all addresses within the address space are valid, it is the responsibility of the user to check that the type of access is valid.

- Example: the contents of the flash memory can be used as a source operand, but can only be written to in certain conditions.

Instruction format I: Double operand

❑ Organization of instructions with two operands:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Op-code				S-Reg				Ad	B/W	As		D-Reg			

❑ Double operand instructions:

Mnemonic	Operation	Description
Arithmetic instructions		
ADD(.B or .W) src,dst	src+dst→dst	Add source to destination
ADDC(.B or .W) src,dst	src+dst+C→dst	Add source and carry to destination
DADD(.B or .W) src,dst	src+dst+C→dst (dec)	Decimal add source and carry to destination
SUB(.B or .W) src,dst	dst+.not.src+1→dst	Subtract source from destination
SUBC(.B or .W) src,dst	dst+.not.src+C→dst	Subtract source and not carry from destination
Logical and register control instructions		
AND(.B or .W) src,dst	src.and.dst→dst	AND source with destination
BIC(.B or .W) src,dst	.not.src.and.dst→dst	Clear bits in destination
BIS(.B or .W) src,dst	src.or.dst→dst	Set bits in destination
BIT(.B or .W) src,dst	src.and.dst	Test bits in destination
XOR(.B or .W) src,dst	src.xor.dst→dst	XOR source with destination
Data instructions		
CMP(.B or .W) src,dst	dst-src	Compare source to destination
MOV(.B or .W) src,dst	src→dst	Move source to destination

Examples: double operand (1/3)

- **Move the contents of register R5 to register R4:**

```
MOV R5,R4
```

  - Instruction code: 0x4504

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 0	0 1 0 1	0	0	0 0	0 1 0 0
MOV	R5	Register	16-Bits	Register	R4

  - This instruction uses 1 word;

  - The instruction coding specifies that the CPU must perform a 16-bit data `MOV` instruction, with the contents of register R5 as the source and with register R4 as the destination.

Examples: double operand (2/3)

- **Move the contents of register R5 to the address in memory TONI:**

```
MOV R5,TONI
```

  - Instruction code: 0x4580

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 0	0 1 0 1	1	0	0 0	0 0 0 0
MOV	R5	Symbolic	16 Bits	Register	PC

  - This instruction uses 2 words;
  - The instruction coding specifies that the CPU must perform a 16-bit data `MOV` instruction using the contents of register R5 to the memory address pointed to by `X1 + PC`;
  - The word `X1` is stored in the word following the instruction.

Examples: double operand (3/3)

- **Move the contents between the memory addresses EDEN and TONI:**

```
MOV EDEN,TONI
```

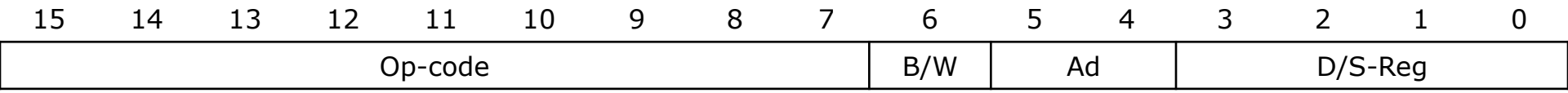
  - Instruction code: 0x4090

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 0	0 0 0 0	1	0	0 1	0 0 0 0
MOV	PC	Symbolic	16-Bits	Symbolic	PC

  - This instruction uses 3 words;
  - The instruction coding specifies that the CPU must perform a 16-bit data `MOV` instruction using the contents of the EDEN memory address pointed to by `X1 + PC` to the TONI memory address pointed to by `X2 + PC`;
  - The word `X1` followed by the word `X2` are stored after the instruction.

Instruction format II: Single operand

- The instructions with one operand are coded using the following structure:



- Single operand instructions:

Mnemonic	Operation	Description
Logical and register control instructions		
`RRA(.B or .W) dst`	MSB→MSB→…LSB→C	Roll destination right
`RRC(.B or .W) dst`	C→MSB→…LSB→C	Roll destination right through (from) carry
`SWPB( or .W) dst`	Swap bytes	Swap bytes in destination
`SXT dst`	bit 7→bit 8…bit 15	Sign extend destination
`PUSH(.B or .W) src`	SP-2→SP, src→@SP	Push source on stack
Program flow control instructions		
`CALL(.B or .W) dst`	SP-2→SP, PC+2→@SP dst→PC	Subroutine call to destination
`RETI`	TOS→SR, SP+2→SP TOS→PC, SP+2→SP	Return from interrupt

Examples: single operand (1/2)

- **Move the contents of register R5 to the right with carry flag:**

```
RRC     R5
```

  - Instruction code: 0x1005

Op-code	B/W	Ad	D-reg
0 0 0 1 0 0 0 0 0	0	0 0	0 1 0 1
RRC	16 bits	Register	R5

  - This instruction uses 1 word;
  - The instruction coding specifies that the CPU must perform a 16-bit data `RRC` instruction using the contents of register R5.

Examples: single operand (2/2)

- **Rotate the contents of memory TONI to the right with carry flag:**

```
RRC    TONI
```

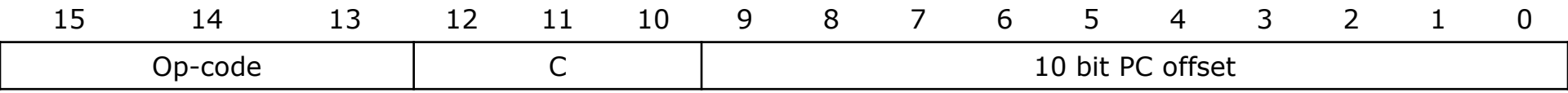
  - Instruction code: 0x1010

Op-code	B/W	Ad	D-reg
0 0 0 1 0 0 0 0 0	0	0 1	0 0 0 0
RRC	16 bits	Symbolic	PC

  - This instruction uses 2 words;
  - The instruction coding specifies that the CPU must perform a 16-bit data `RRC` instruction pointed to by `X1 + PC`;
  - Word `X1` is stored in the word following the instruction.

Jump instructions format (1/3)

- **These instructions are used to direct program flow to another part of the program.**



Mnemonic	Description
Program flow control instructions	
JEQ/JZ label	Jump to label if zero flag is set
JNE/JNZ label	Jump to label if zero flag is reset
JC label	Jump to label if carry flag is set
JNC label	Jump to label if carry flag is reset
JN label	Jump to label if negative flag is set
JGE label	Jump to label if greater than or equal
JL label	Jump to label if less than
JMP label	Jump to label unconditionally

Jump instructions format (2/3)

- **The op-code always takes the value 001b, indicating that it is a jump instruction;**

- **The condition on which a jump occurs depends on the C field consisting of 3 bits, and may take the following values:**
  - 000b: jump if not equal;
  - 001b: jump if equal;
  - 010b: jump if carry flag equal to zero;
  - 011b: jump if carry flag equal to one;
  - 100b: jump if negative (N = 1);
  - 101b: jump if greater than or equal (N=V or (N OR V=0));
  - 110b: jump if lower (N! = V or (V XOR N = 1));
  - 111b: unconditional jump.

Jump instructions format (3/3)

- The jumps are executed based on the PC contents, are controlled by the status bits, but do not affect the status bits;
- The jump off-set is represented by a signed 10-bit value:

$$PC_{new} = PC_{old} + 2 + PC_{offset} \times 2$$

- The range of the jump can be between -511 to +512 words, in relation to the PC position.

Examples: jump format (1/2)

- **Continue execution at the label `main` if the carry flag is active:**

```
JC     main
```

  - Instruction code: 0x2FE4

Op-code	C	10-Bit PC offset
0 0 1	0 1 1	1 1 1 1 1 0 0 1 0 0
JC	Carry = 1	– 0x1C

  - This instruction uses 1 word;
  - The instruction coding specifies that the PC must be loaded with the value resulting from the offset - 0x1C being applied to the previous expression.

Examples: jump format (2/2)

- **Continue to unconditionally execute code at the label**

```
main:
JMP    main
```

  - Instruction code: 0x3FE3

Op-code	C	10-Bit PC offset
0 0 1	1 1 1	1 1 1 1 1 0 0 0 1 1
JMP	unconditional	– 0x1D

  - This instruction uses 1 word;
  - The instruction coding specifies that the PC must be loaded with the value resulting from the offset - 0x1D being applied to the previous expression.

Emulated instructions (1/3)

- **In addition to the 27 CPU instructions, the following 24 emulated instructions can also be used:**

Mnemonic	Operation	Emulation	Description
Arithmetic instructions			
ADC(.B or .W) dst	dst+C→dst	ADDC(.B or .W) #0,dst	Add carry to destination
DADC(.B or .W) dst	dst+C→dst (decimally)	DADD(.B or .W) #0,dst	Decimal add carry to destination
DEC(.B or .W) dst	dst-1→dst	SUB(.B or .W) #1,dst	Decrement destination
DECD(.B or .W) dst	dst-2→dst	SUB(.B or .W) #2,dst	Decrement destination twice
INC(.B or .W) dst	dst+1→dst	ADD(.B or .W) #1,dst	Increment destination
INCD(.B or .W) dst	dst+2→dst	ADD(.B or .W) #2,dst	Increment destination twice
SBC(.B or .W) dst	dst+0FFFFh+C→dst dst+0FFh→dst	SUBC(.B or .W) #0,dst	Subtract source and borrow /.NOT. carry from dest.

Emulated instructions (2/3)

- **In addition to the 27 CPU instructions, the following 24 emulated instructions can also be used (continued):**

Mnemonic	Operation	Emulation	Description
Logical and register control instructions			
INV(.B or .W) dst	.NOT.dst→dst	XOR(.B or .W) #0(FF)FFh,dst	Invert bits in destination
RLA(.B or .W) dst	C←MSB←MSB-1…LSB+1←LSB←0	ADD(.B or .W) dst,dst	Rotate left arithmetically
RLC(.B or .W) dst	C←MSB←MSB-1…LSB+1←LSB←C	ADDC(.B or .W) dst,dst	Rotate left through carry
Program flow control			
BR dst	dst→PC	MOV dst,PC	Branch to destination
DINT	0→GIE	BIC #8,SR	Disable (general) interrupts
EINT	1→GIE	BIS #8,SR	Enable (general) interrupts
NOP	None	MOV #0,R3	No operation
RET	@SP→PC SP+2→SP	MOV @SP+,PC	Return from subroutine

Emulated instructions (3/3)

- **In addition to the 27 CPU instructions, the following 24 emulated instructions can also be used (continued):**

Mnemonic	Operation	Emulation	Description
Data instructions			
CLR(.B or .W) dst	0→dst	MOV(.B or .W) #0,dst	Clear destination
CLRC	0→C	BIC #1,SR	Clear carry flag
CLRN	0→N	BIC #4,SR	Clear negative flag
CLRZ	0→Z	BIC #2,SR	Clear zero flag
POP(.B or .W) dst	@SP→temp SP+2→SP temp→dst	MOV(.B or .W) @SP+,dst	Pop byte/word from stack to destination
SETC	1→C	BIS #1,SR	Set carry flag
SETN	1→N	BIS #4,SR	Set negative flag
SETZ	1→Z	BIS #2,SR	Set zero flag
TST(.B or .W) dst	dst + 0FFFFh + 1 dst + 0FFh + 1	CMP(.B or .W) #0,dst	Test destination

Examples: Emulated instructions (1/6)

- **Clear the contents of register R5:**

```
CLR R5
```

  - Instruction code: 0x4305

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 0	0 0 1 1	0	0	0 0	0 1 0 1
MOV	R3	Register	16 Bits	Register	R5

  - This instruction is equivalent to using `MOV R3, R5` where R3 takes the value `#0`.

Examples: Emulated instructions (2/6)

- **Increment the content of register R5:**

```
INC   R5
```

  - Instruction code: 0x5315

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 1	0 0 1 1	0	0	0 1	0 1 0 1
ADD	R3	Register	16 Bits	Indexed	R5

  - This instruction is equivalent to having `ADD 0(R3),R5` where R3 takes the value `#1`.

Examples: Emulated instructions (3/6)

- **Decrement the contents of register R5:**

```
DEC   R5
```

  - Instruction code: 0x8315

Op-code	S-reg	Ad	B/W	As	D-reg
1 0 0 0	0 0 1 1	0	0	0 1	0 1 0 1
SUB	R3	Register	16 Bits	Indexed	R5

  - This instruction is equivalent to using `SUB 0(R3),R5` where R3 takes the value `#1`.

Examples: Emulated instructions (4/6)

- **Decrement by two the contents of register R5:**

```
DECD  R5
```

  - Instruction code: 0x8325

Op-code	S-reg	Ad	B/W	As	D-reg
1 0 0 0	0 0 1 1	0	0	1 0	0 1 0 1
SUB	R3	Register	16 Bits	Indirect	R5

  - This instruction is equivalent to using `SUB @R3,R5` where R3 points to the value `#2`.

Examples: Emulated instructions (5/6)

- **Do not carry out any operation:**

  `NOP`

  - Instruction code: 0x4303

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 0	0 0 1 1	0	0	0 0	0 0 1 1
MOV	R3	Register	16 Bits	Register	R3

  - This instruction is equivalent to using `MOV R3,R3` and therefore the contents of R3 are moved to itself.

Examples: Emulated instructions (6/6)

- **Add the carry flag to the register R5:**

```
ADC   R5
```

  - Instruction code: 0x6305

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 1 0	0 0 1 1	0	0	0 0	0 1 0 1
ADDC	R3	Register	16 Bits	Register	R5

  - This instruction is equivalent to using `ADDC R3,R5` where R3 takes the value `#0`.

Addressing modes (1/2)

- There are seven addressing modes that take a source operand and four of these addressing modes also take a destination operand;
- The operands can be located in any memory space address, but the user needs to be aware of the effects that their accesses may have;
- The addressing modes are selected by the `As` and `Ad` fields that make up the instruction;
- The following table summarizes the seven addressing modes:

Addressing modes (2/2)

Operands (single-operand instructions) Source operands (double-operand instructions)			Destination operands (double-operand instructions)		
Addressing mode	As	S-reg	Addressing mode	Ad	D-reg
Register mode	0 0	0 0 0 0 to 1 1 1 1	Register mode	0	0 0 0 0 to 1 1 1 1
Indexed mode	0 1	0 0 0 1, 0 0 1 1 to 1 1 1 1	Indexed mode	1	0 0 0 1, 0 0 1 1 to 1 1 1 1
Symbolic mode	0 1	0 0 0 0	Symbolic mode	1	0 0 0 0
Absolute mode	0 1	0 0 1 0	Absolute mode	1	0 0 1 0
Indirect register mode	1 0	0 0 0 0 to 1 1 1 1			
Indirect auto increment mode	1 1	0 0 0 1 to 1 1 1 1			
Immediate mode	1 1	0 0 0 0			

Register mode (1/3)

- **In Register addressing mode, the contents of a register is used as the operand;**
- **This type of addressing mode can be used both with source and destination operands.**
- **Move the contents of register R5 to register R4:**

```
MOV R5,R4
```

  - Instruction code: 0x4504

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 0	0 1 0 1	0	0	0 0	0 1 0 0
MOV	R5	Register	16-bit	Register	R4

Register mode (2/3)

- **Move the contents of register R5 to register R4**

```
MOV R5,R4
```

  - Instruction code: 0x4504

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 0	0 1 0 1	0	0	0 0	0 1 0 0
MOV	R5	Register	16-bit	Register	R4

  - The 16-bit contents (`B/W = 0`) of register R5 (S-reg) are transferred to the register R4 (D-reg);
  - After the instruction fetch, the PC is incremented by 2 and points to the next instruction to be executed;
  - The addressing mode used for the source and destination operands is determined by `As = 00` (Register mode) and `Ad = 0` (Register mode), respectively.

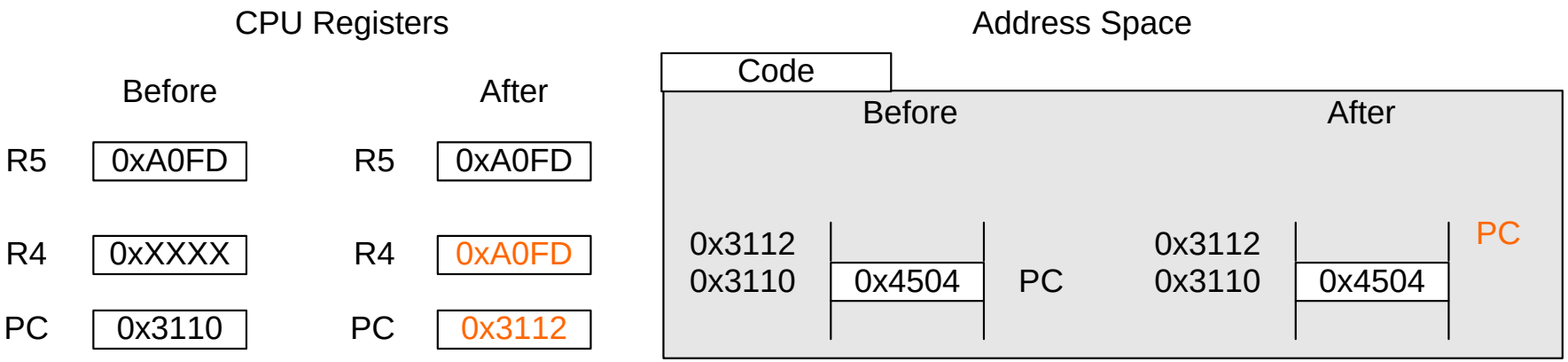
Register mode (3/3)

- **Move the contents of register R5 to register R4 (cont.):**

```
MOV R5,R4
```

  - Instruction code: 0x4504

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 0	0 1 0 1	0	0	0 0	0 1 0 0
MOV	R5	Register	16-bit	Register	R4



Indexed mode (1/4)

- In indexed mode, whether used to indicate the source address or the destination address, the sum of the register contents and the signed offset point to the location in memory;
- The offset value is stored in the word following the instruction;
- After the execution of the instruction, the contents of registers are not affected and the PC is incremented to point to the next instruction to be executed;
- This addressing mode is useful to access data stored in tables. Apart from the registers PC and SR, all other registers can be used as an index in indexed mode.

Indexed mode (2/4)

- **Move the byte pointed to by (R5 + 4) to the byte pointed to by (R4 + 1):**

```
MOV.B 4(R5),1(R4)
```

  - Instruction code: 0x45D4

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 0	0 1 0 1	1	1	0 1	0 1 0 0
MOV	R5	Indexed	8-bit	Indexed	R4

  - The instruction coding specifies that the byte (`B/W = 1`) pointed to by the sum of the contents of register R5 (`S-reg = 0101`) and the word `X1` should be moved to the memory address pointed to by the sum of register R4 (`D-reg = 0100`) and the contents of the word `X2`;

Indexed mode (3/4)

- **Move the byte pointed to by (R5 + 4) to the byte pointed to by (R4 + 1) (continued):**

```
MOV.B 4(R5),1(R4)
```

  - Instruction code: 0x45D4

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 0	0 1 0 1	1	1	0 1	0 1 0 0
MOV	R5	Indexed	8-bit	Indexed	R4

  - The words `X1` and `X2` are located in the memory addresses following the instruction;
  - The addressing mode used for the source and destination operands is controlled by the bits `Ad = 1` (Indexed mode) and `As = 01` (Indexed mode), because (`D-reg = 0100`) and (`S-reg = 0101`) respectively.

Indexed mode (4/4)

- **Move the byte pointed to by (R5 + 4) to the byte pointed to by (R4 + 1) (continued):**

```
MOV.B 4(R5),1(R4)
```

CPU Registers

	Before		After
R5	0x0200	R5	0x0200
R4	0x0200	R4	0x0200
PC	0x3110	PC	0x3116

Destination Address

0x0200 (R4)
+ 0x0001 (X2)
0x0201

Source Address

0x0200 (R5)
+ 0x0004 (X1)
0x0204

Address Space

Code

	Before			After		
0x3116				0x3116		PC
0x3114	0x0001	X2		0x3114	0x0001	X2
0x3112	0x0004	X1		0x3112	0x0004	X1
0x3110	0x45D4	PC		0x3110	0x45D4	

Data

0x0206				0x0206		
0x0204	0x9ABC			0x0204	0x9ABC	
0x0202	0x5678			0x0202	0x5678	
0x0200	0x1234	1(R4)		0x0200	0x129A	1(R4)
0x0206				0x0206		
0x0204	0x9ABC	4(R5)		0x0204	0x9ABC	4(R5)
0x0202	0x5678			0x0202	0x5678	
0x0200	0x1234			0x0200	0x129A	

Symbolic mode (1/5)

- In symbolic addressing mode, for either the source or destination operands, the memory address is determined by adding an offset to the program counter (PC) register;
- The offset value is obtained by determining the code position in the memory, then calculating the difference between the code position and the memory position that should be achieved;
- The assembler determines the offset value and puts it in the word following to the instruction;
- After the execution of the instruction, the program counter (PC) register is incremented in order to point to the next instruction.

Symbolic mode (2/5)

- Although this mode of address is similar to register mode and also requests more cycles, but in this case, the program counter (PC) is used to point to the data;

- This addressing mode can be used to determine either the source or the destination of the data.

Symbolic mode (3/5)

- **Move the word pointed to by EDEN to the word pointed to by TONI:**

```
MOV EDEN,TONI
```

  - Instruction code: 0x4090

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 0	0 0 0 0	1	0	0 1	0 0 0 0
MOV	PC	Symbolic	16-bit	Symbolic	PC

  - The instruction coding specifies that the value pointed to by the sum of the PC register contents (`S-reg = 0`) and the word `X1` should be moved to the memory address pointed to by the sum of register PC contents (`D-reg = 0`) and the word `X2`;
  - The words `X1` and `X2` are stored in the memory addresses following the instruction.

Symbolic mode (4/5)

- **Move the word pointed to by EDEN to the word pointed to by TONI (continued):**

```
MOV EDEN,TONI
```

  - Instruction code: 0x4090

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 0	0 0 0 0	1	0	0 1	0 0 0 0
MOV	PC	Symbolic	16-bit	Symbolic	PC

  - The addressing mode used for the source and destination operands is controlled by the bits `Ad = 1` (Symbolic mode) and `As = 01` (Symbolic mode) because (`D-reg = 0000`) and (`S-reg = 0000`), respectively.

Symbolic mode (5/5)

- **Move the word pointed to by EDEN to the word pointed to by TONI:**

```
MOV EDEN,TONI
```

CPU Registers

	Before		After
PC	0x3110	PC	0x3116

Destination Address

```
  0x3114
+0xD0EE   (PC)
  0x0202  (X2)
```

Source Address

```
  0x3112
+0xD0EE   (PC)
  0x0200  (X1)
```

Address Space

Code

	Before		After	
0x3116			0x3116	PC
0x3114	0xD0EE	X2	0x3114	0xD0EE X2
0x3112	0xD0EE	X1	0x3112	0xD0EE X1
0x3110	0x4090	PC	0x3110	0x4090

Data

0x0206			0x0206	
0x0204	0x9ABC		0x0204	0x9ABC
0x0202	0x5678	TONI	0x0202	0x1234 TONI
0x0200	0x1234		0x0200	0x1234
0x0206			0x0206	
0x0204	0x9ABC		0x0204	0x9ABC
0x0202	0x5678		0x0202	0x1234
0x0200	0x1234	EDEN	0x0200	0x1234 EDEN

Absolute mode (1/4)

- In Absolute mode, the data memory address is placed after the instruction;
- The difference between this addressing mode and Indexed mode is that the register R2 is now used as an index, using the constants generator to generate the value zero;
- This addressing mode can be used to define either the source address or the destination address.

Absolute mode (2/4)

- **Move the word pointed to by EDEN to the word pointed to by TONI:**

```
MOV &EDEN,&TONI
```

  - Instruction code: 0x4292

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 0	0 0 1 0	1	0	0 1	0 0 1 0
MOV	R2/CG1	Absolute	16-bit	Absolute	R2/CG1

  - From the instruction coding it can be seen that the register R2 has (`S-reg = 0010`) and (`D-reg = 0010`). Register R2 is used as an addresses index, in which the constant generator loads the value zero;
  - When the contents of this register is added to the offset value `X1` or `X2`, the source and destination addresses are obtained;
  - The words `X1` and `X2` are stored in the memory addresses following the instruction.

Absolute mode (3/4)

- **Move the word pointed to by EDEN to the word pointed to by TONI (continued):**

```
MOV &EDEN,&TONI
```

  - Instruction code: 0x4292

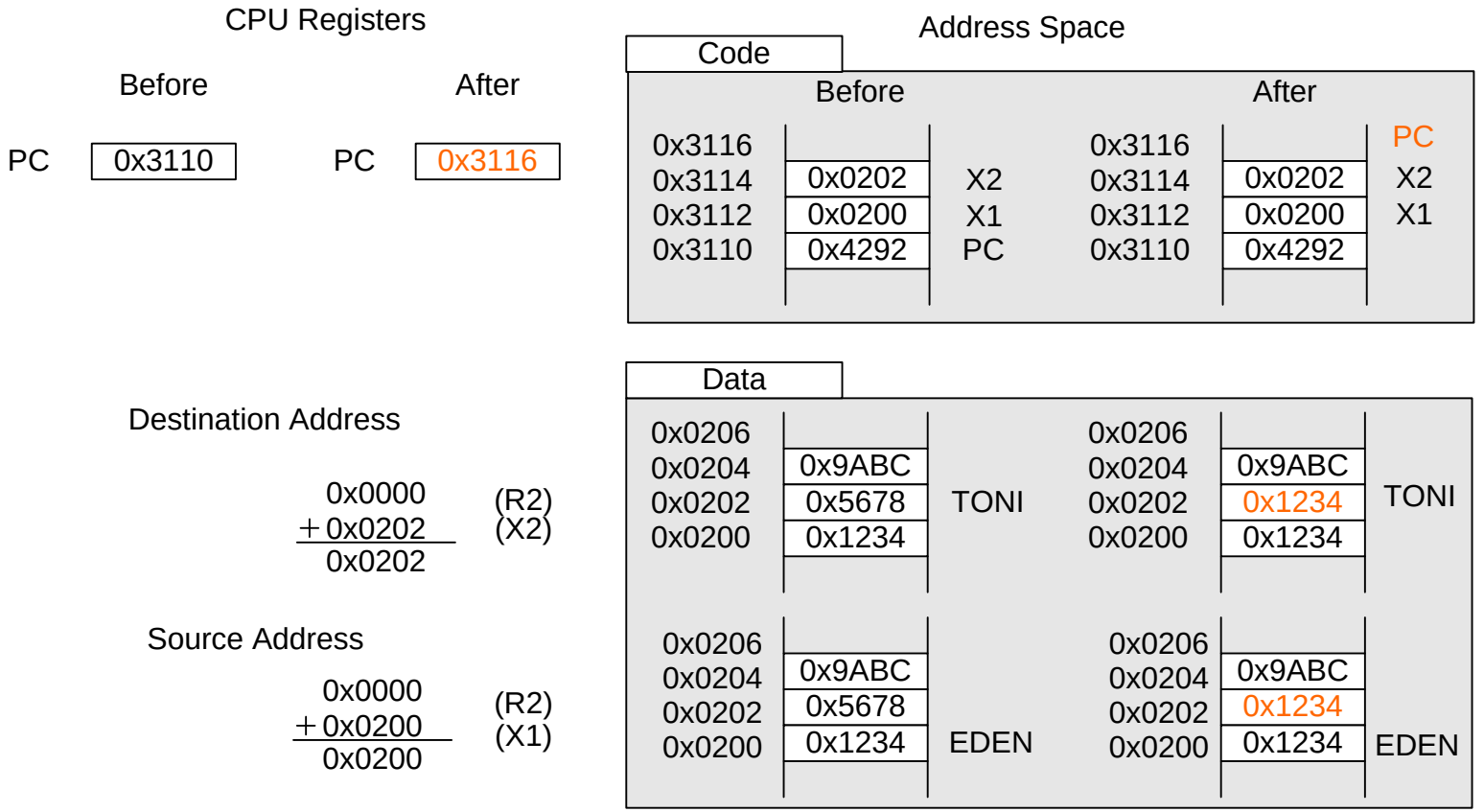
Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 0	0 0 1 0	1	0	0 1	0 0 1 0
MOV	R2/CG1	Absolute	16-bit	Absolute	R2/CG1

  - The addressing mode used for the source and destination operands is specified by the bits `Ad = 1` (Absolute mode) and `As = 01` (Absolute mode) because (`D-reg = 0010`) and (`S-reg = 0010`), respectively.

Absolute mode (4/4)

- **Move the word pointed to by EDEN to the word pointed to by TONI (continued):**

```
MOV &EDEN,&TONI
```



Indirect register mode (1/3)

- In Indirect register mode, any of the 16 CPU registers can be used. If R2 or R3 are used then a constant value is used as an operand, #0x04 for R2 and #0x02 for R3;
- A restriction arises from the fact that this addressing mode can only be used to specify the source operand address in dual-operand instructions;
- An alternative way to avoid this restriction is to use indexed mode to specify the destination operand address, with a zero offset.

Indirect register mode (2/3)

- **Move the word pointed to by R5 to the word pointed to by R4:**

```
MOV @R5,0(R4)
```

  - Instruction code: 0x45A4

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 0	0 1 0 1	1	0	1 0	0 1 0 0
MOV	R5	Indexed	16-bit	Indirect	R4

  - The instruction coding specifies that register R5 (`S-reg = 0101`) uses the source address (`As = 10`);

  - The destination address is pointed to in Indexed mode (`Ad = 1`) by R4 (`D-reg = 0100`), using a zero value offset.

Indirect register mode (3/3)

- **Move the word pointed to by R5 to the word pointed to by R4 (continued):**

```
MOV @R5,0(R4)
```

CPU Registers

	Before		After
R5	0x0200	R5	0x0200
R4	0x0202	R4	0x0202
PC	0x3110	PC	0x3114

Destination Address

```
  0x0202   (R4)
+ 0x0000   (X1)
  ------
  0x0202
```

Source Address

Address Space

Code

	Before			After		
0x3114				0x3114		PC
0x3112	0x0000	X1		0x3112	0x0000	X1
0x3110	0x45A4	PC		0x3110	0x45A4	

Data

	Before			After		
0x0206				0x0206		
0x0204	0x9ABC			0x0204	0x9ABC	
0x0202	0x5678	0(R4)		0x0202	0x1234	0(R4)
0x0200	0x1234			0x0200	0x1234	
0x0206				0x0206		
0x0204	0x9ABC			0x0204	0x9ABC	
0x0202	0x5678			0x0202	0x1234	
0x0200	0x1234	@R5		0x0200	0x1234	@R5

Indirect auto-increment mode (1/3)

- This addressing mode is similar to the previous one;
- The contents of source register is incremented according to the data type processed;
- If the data value is byte, the source register is incremented by 1;
- If the data value is a word, the source register is incremented by 2;
- Note that this addressing mode can only be used to specify the *source* operand in dual-operand instructions.

Indirect auto-increment mode (2/3)

- **Move the word pointed to by R5 to the word pointed to by R4, and increment the source pointer:**

  `MOV @R5+,0(R4)`

  - Instruction code: 0x45B4

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 0	0 1 0 1	1	0	1 1	0 1 0 0
MOV	R5	Indexed	16-bit	Ind. aut. inc.	R4

  - The instruction coding specifies that the register R5 (`S-reg = 0101`) contains the source address (`As = 11`);
  - The destination address is pointed to in indexed mode by R4 (`D-reg = 0100`), using a zero value offset;
  - The execution of the instruction increments the contents of register R5 by 2.

Indirect auto-increment mode (3/3)

- **Move the word pointed to by R5 to the word pointed to by R4, and increment the source pointer (continued):**

```
MOV @R5+,0(R4)
```

CPU Registers

	Before		After
R5	0x0200	R5	0x0202
R4	0x0202	R4	0x0202
PC	0x3110	PC	0x3114

Destination Address

```
  0x0202   (R4)
+ 0x0000   (X1)
  ------
  0x0202
```

Source Address

Address Space

Code

	Before			After		
0x3114			0x3114		PC
0x3112	0x0000	X1	0x3112	0x0000	X1
0x3110	0x45A4	PC	0x3110	0x45A4	

Data

	Before			After		
0x0206			0x0206		
0x0204	0x9ABC		0x0204	0x9ABC	
0x0202	0x5678	0(R4)	0x0202	0x1234	0(R4)
0x0200	0x1234		0x0200	0x1234	
0x0206			0x0206		
0x0204	0x9ABC		0x0204	0x9ABC	
0x0202	0x5678		0x0202	0x1234	
0x0200	0x1234	@R5	0x0200	0x1234	@R5

Immediate mode (1/2)

- **The Immediate addressing mode allows values to be loaded directly into registers or memory addresses. It can only be used with source operands.**

- **Move the value 0x0200 to R5:**

```
MOV #0x0200,R5
```

  - Instruction code: 0x4035

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 0	0 0 0 0	0	0	1 1	0 1 0 1
MOV	PC	Register	16-bit	Immediate	R5

  - The instruction coding specifies that the register PC (`S-reg = 0000`) is used to define the address of the word in memory that loads the register R5 (`D-reg = 0101`) with (`As = 11`).

Immediate mode (2/2)

- **Move the value 0x0200 to R5 (continued):**

```
MOV 0x0200,R5
```

CPU Registers

	Before		After
R5	0xXXXX	R5	0x0200
PC	0x3110	PC	0x3114

Address Space

Code

Before			After		
0x3114			0x3114		PC
0x3112	0x0200	X1	0x3112	0x0200	X1
0x3110	0x4035	PC	0x3110	0x4035	