Instruction Set Architecture

EE3376

–Adapted from notes from BYU ECE124

Topics to Cover…

- MSP430 ISA
- MSP430 Registers, ALU, Memory
- Instruction Formats
- Addressing Modes
- Double Operand Instructions
- Single Operand Instructions
- Jump Instructions
- Emulated Instructions
  - http://en.wikipedia.org/wiki/TI_MSP430

–Adapted from notes from BYU ECE124

Levels of Transformation

–Problems

–Algorithms

– C Instructions

–Language (Program) –*Programmable*

–Assembly Language

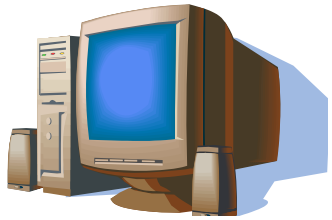
– MSP 430 ISA

–Machine (ISA) Architecture –*Computer Specific*

–Microarchitecture –*Manufacturer Specific*

–Circuits

–Devices



–Adapted from notes from BYU ECE124

Instruction Set Architecture

- The computer ISA defines all of the *programmer-visible* components and operations of the computer
  - memory organization
    - address space -- how may locations can be addressed?
    - addressibility -- how many bits per location?
  - register set (a place to store a collection of bits)
    - how many? what size? how are they used?
  - instruction set
    - Opcodes (operation selection codes)
    - data types (data types: byte or word)
    - addressing modes (coding schemes to access data)
- ISA provides all information needed for someone that wants to write a program in machine language (or translate from a high-level language to machine language).

–Adapted from notes from BYU ECE124

MSP430 Instruction Set Architecture

- MSP430 CPU specifically designed to allow the use of modern programming techniques, such as:
  - the computation of jump addresses
  - data processing in tables
  - use of high-level languages such as C.
- 64KB memory space with 16 16-bit registers that reduce fetches to memory.
- Implements RISC architecture with 27 instructions and 7 addressing modes.

–Adapted from notes from BYU ECE124

MSP430 16-bit RISC

- Orthogonal architecture with every instruction usable with every addressing mode.
- Full register access including program counter, status registers, and stack pointer.
- Single-cycle register operations.
- 16-bit address bus allows direct access and branching throughout entire memory range.
- 16-bit data bus allows direct manipulation of word-wide arguments.
- Word and byte addressing and instruction formats.

–Adapted from notes from BYU ECE124

MSP430 Registers

- The MSP430 CPU has 16 registers
  - Large 16-bit register file eliminates single accumulator bottleneck
  - High-bandwidth 16-bit data and address bus
- 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 by 2, 4, or 6 to point to the next instruction.

–Adapted from notes from BYU ECE124

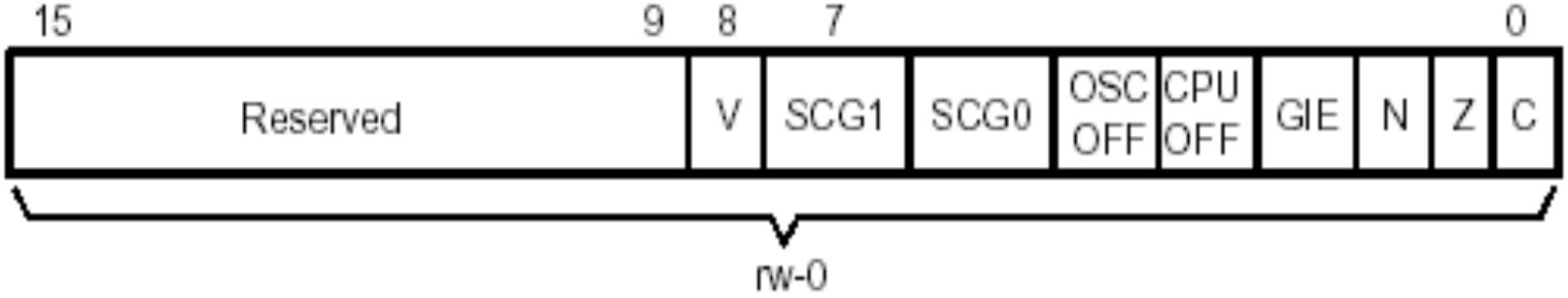
MSP430 Registers

- R1 (SP) – Stack Pointer
  - The MSP430 CPU stores the return address of routines or interrupts on the stack
  - User programs store local data on the stack
  - The SP can be incremented or decremented automatically with each stack access
  - The stack “grows down” thru RAM and thus SP must be initialized with a valid RAM address
  - SP always points to an even address, so its LSB is always zero

–Adapted from notes from BYU ECE124

MSP430 Registers

- R2 (SR/CG1) – Status Register
  - The status of the MSP430 CPU is defined by a set of bits contained in register R2
  - This register can only be accessed through register addressing mode - all other addressing modes are reserved to support the constants generator
  - The status register is used for clock selection, interrupt enable/disable, and instruction result status



–Adapted from notes from BYU ECE124

R2 (SR) – Status Register

V	Overflow bit – set when arithmetic operation overflows the signed-variable range.
SCG1	System clock generator 1 – turns off the SMCLK.
SCG0	System clock generator 0 – turns off the DCO dc generator.
OSCOFF	Oscillator off – turns off the LFXT1 crystal oscillator.
CPUOFF	CPU off – turns off the CPU.
GIE	General interrupt enable – enables maskable interrupts.
N	Negative bit – set when the result of a byte or word operation is negative.
Z	Zero bit – set when the result of a byte or word operation is 0.
C	Carry bit – set when the result of a byte or word operation produces a carry.

–Adapted from notes from BYU ECE124

R2 (SR) – Status Register

- R2 (SR/CG1), R3 (CG2) – Constant Generators
  - Six different constants commonly used in programming can be generated using the registers R2 and R3, without adding a 16-bit extension word of code to the instruction

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

–Adapted from notes from BYU ECE124

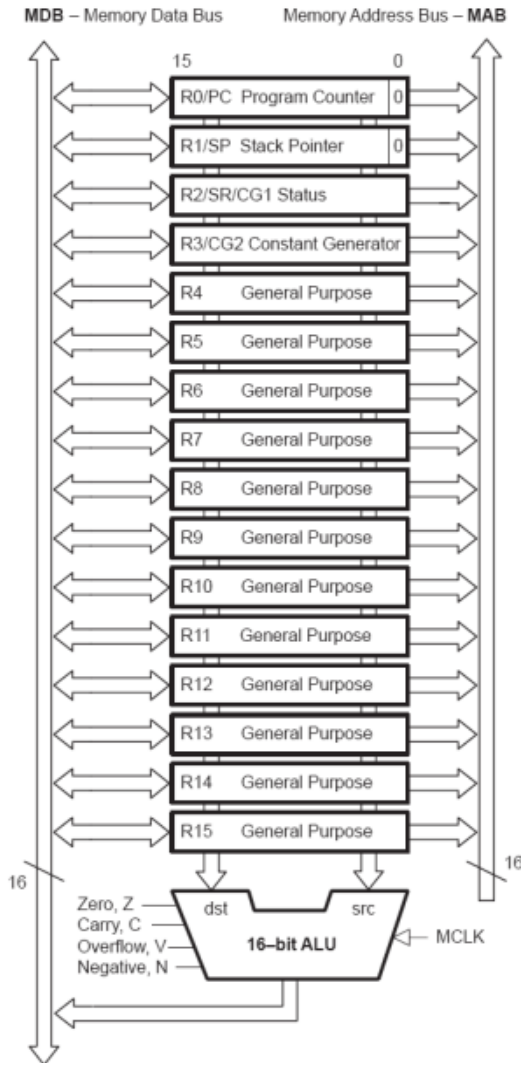
MSP430 Registers

- 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
  - Instruction formats support byte or word accesses
  - The status bits of the CPU in the SR are updated after the execution of a register instruction.

–Adapted from notes from BYU ECE124

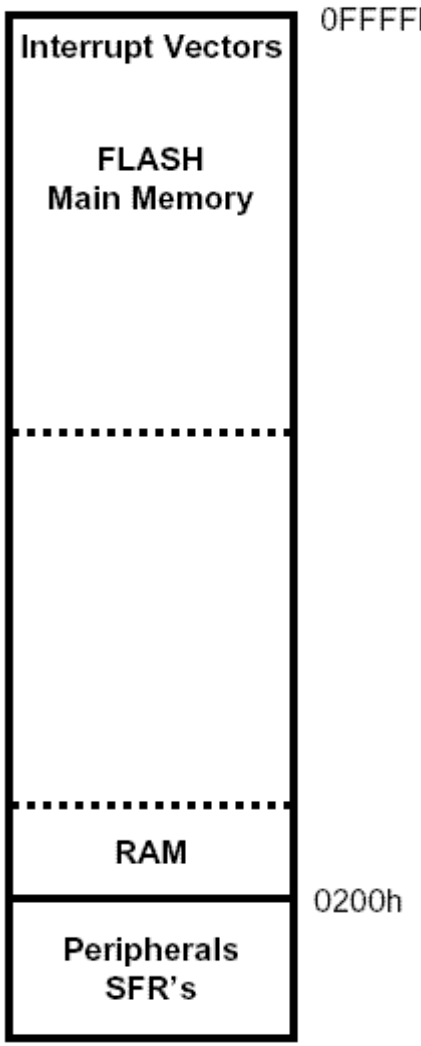
MSP430 ALU

- 16 bit Arithmetic Logic Unit (ALU).
  - Performs instruction arithmetic and logical operations
  - Instruction execution affects the state of the following flags:
    - Zero (Z)
    - Carry (C)
    - Overflow (V)
    - Negative (N)
  - The MCLK (Master) clock signal drives the CPU.



–Adapted from notes from BYU ECE124

MSP430 Memory



- Unified 64KB continuous memory map
- Same instructions for data and peripherals
- Program and data in Flash or RAM with no restrictions
- Designed for modern programming techniques such as pointers and fast look-up tables

–Adapted from notes from BYU ECE124

Anatomy of an Instruction

- Opcode
  - What the instruction does – verb
  - May or may not require operands – objects
- Source Operand
  - 1st data object manipulated by the instruction
- Destination Operand
  - 2nd data object manipulated by the instruction
  - Also where results of operation are stored.
- Addressing Modes

–Adapted from notes from BYU ECE124

Instruction Format

- There are three formats used to encode instructions for processing by the CPU core
  - Double operand
  - Single operand
  - Jumps
- 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

–Adapted from notes from BYU ECE124

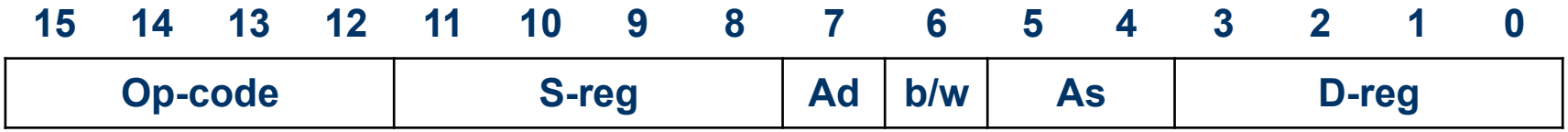
Instruction Format

- 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.

–Adapted from notes from BYU ECE124

MPS430 Instruction Formats

- Format I: Instructions with two operands:



- Format II: Instruction with one operand:

- Format II: Jump instructions:

–Adapted from notes from BYU ECE124

3 Instruction Formats

```
; Format I Source and Destination
```

Op-Code	Source-Register	Ad	B/W	As	Destination-Register

```
5405            add.w   R4,R5                   ; R4+R5=R5  xxxx
5445            add.b   R4,R5                   ; R4+R5=R5  00xx

; Format II Destination Only
```

Op-Code	B/W	Ad	D/S- Register

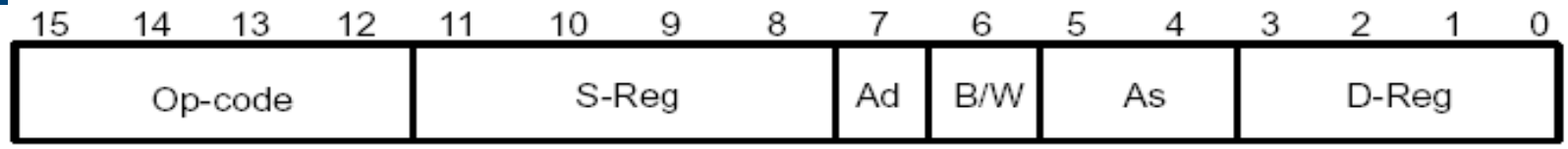
```
6404            rlc.w   R4                      ;
6444            rlc.b   R4                      ;

; Format III There are 8(Un)conditional Jumps
```

Op-Code	Condition	10-bit PC offset

```
3c28            jmp     Loop_1                  ; Goto Loop_1
```


–Adapted from notes from BYU ECE124

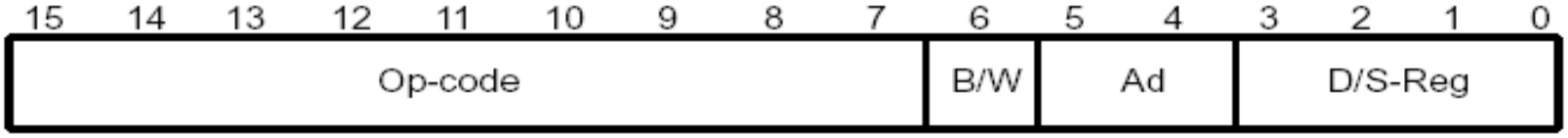
Double Operand Instructions



Mnemonic	S-Reg, D-Reg	Operation	Status Bits			
			V	N	Z	C
MOV(.B)	src,dst	src → dst	–	–	–	–
ADD(.B)	src,dst	src + dst → dst	*	*	*	*
ADDC(.B)	src,dst	src + dst + C → dst	*	*	*	*
SUB(.B)	src,dst	dst + .not.src + 1 → dst	*	*	*	*
SUBC(.B)	src,dst	dst + .not.src + C → dst	*	*	*	*
CMP(.B)	src,dst	dst – src	*	*	*	*
DADD(.B)	src,dst	src + dst + C → dst (decimally)	*	*	*	*
BIT(.B)	src,dst	src .and. dst	0	*	*	*
BIC(.B)	src,dst	.not.src .and. dst → dst	–	–	–	–
BIS(.B)	src,dst	src .or. dst → dst	–	–	–	–
XOR(.B)	src,dst	src .xor. dst → dst	*	*	*	*
AND(.B)	src,dst	src .and. dst → dst	0	*	*	*

–Adapted from notes from BYU ECE124

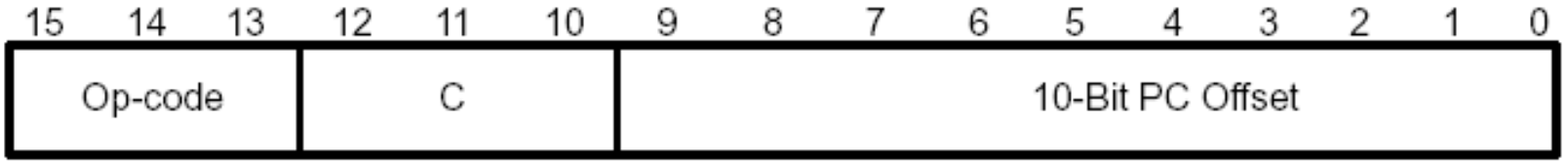
Single Operand Instruction



Mnemonic	S-Reg, D-Reg	Operation	Status Bits V	N	Z	C
RRC(.B)	dst	C → MSB →.....LSB → C	*	*	*	*
RRA(.B)	dst	MSB → MSB →....LSB → C	0	*	*	*
PUSH(.B)	src	SP – 2 → SP, src → @SP	–	–	–	–
SWPB	dst	Swap bytes	–	–	–	–
CALL	dst	SP – 2 → SP, PC+2 → @SP dst → PC	–	–	–	–
RETI		TOS → SR, SP + 2 → SP TOS → PC,SP + 2 → SP	*	*	*	*
SXT	dst	Bit 7 → Bit 8.....Bit 15	0	*	*	*

Adapted from notes from BYU ECE124

Jump Instructions



Mnemonic	S-Reg, D-Reg	Operation
JEQ/JZ	Label	Jump to label if zero bit is set
JNE/JNZ	Label	Jump to label if zero bit is reset
JC	Label	Jump to label if carry bit is set
JNC	Label	Jump to label if carry bit is reset
JN	Label	Jump to label if negative bit is set
JGE	Label	Jump to label if (N .XOR. V) = 0
JL	Label	Jump to label if (N .XOR. V) = 1
JMP	Label	Jump to label unconditionally

–Adapted from notes from BYU ECE124

Source Addressing Modes

- The MSP430 has four basic modes for the source address:
  - **Rs** - Register
  - **x(Rs)** - Indexed Register
  - **@Rs** - Register Indirect
  - **@Rs+** - Indirect Auto-increment
- In combination with registers R0-R3, three additional source addressing modes are available:
  - **label** - PC Relative, **x(PC)**
  - **&label** – Absolute, **x(SR)**
  - **#n** – Immediate, **@PC+**

–Adapted from notes from BYU ECE124

Destination Addressing Modes

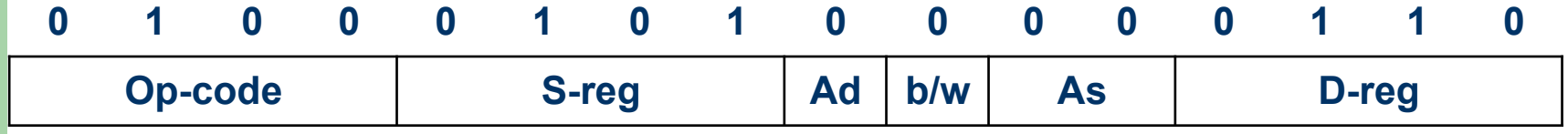
- There are two basic modes for the destination address:
  - **Rd** - Register
  - **x(Rd)** - Indexed Register
- In combination with registers R0/R2, two additional destination addressing modes are available:
  - **label** - PC Relative, **x(PC)**
  - **&label** – Absolute, **x(SR)**

–Adapted from notes from BYU ECE124

Register Mode (Rn)

- The most straightforward addressing mode and is available for both source and destination
  - **Example:**

    mov.w r5,r6 ; move word from r5 to r6
- The registers are specified in the instruction; no further data is needed
- Also the fastest mode and does not require an addition cycle
- Byte instructions use only the lower byte, but clear the upper byte when writing

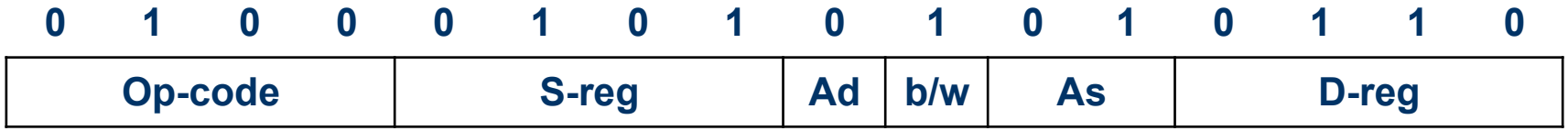


–Adapted from notes from BYU ECE124

Indexed Mode x(Rn)

- The address is formed by adding a constant (index) to the contents of a CPU register
  - **Example:**

    mov.b 3(r5),r6 ; move byte from
    ; $M(3_{10}+r5)$ to r6
- Indexed addressing can be used for source and/or destination, value in r5 is unchanged.
- The index is located in the memory word following the instruction and requires an additional memory cycle
- There is no restriction on the address for a byte, but words must lie on even addresses



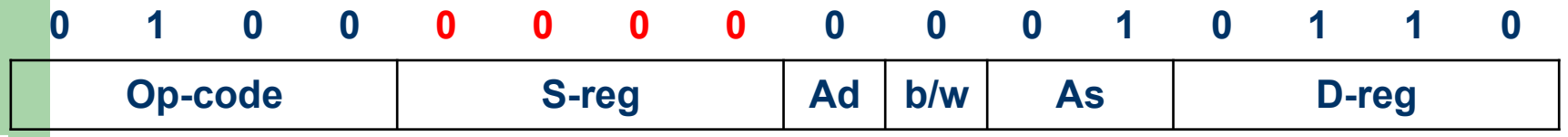
–Adapted from notes from BYU ECE124

Symbolic Mode (PC Relative)

- The address if formed by adding a constant (index) to the program counter (PC)
  - **Example:** (mov.w x(PC), r6 where x=Cnt-PC)

```
mov.w Cnt,r6          ; move word
                      ; M(Cnt+PC) to r6
```

- The PC relative index is calculated by the assembler
- Produces position-independent code, but rarely used in the MSP430 because absolute addressing can reach all memory addresses
- Note: this is NOT an appropriate mode of addressing when referencing fixed locations in memory such as the special function registers (SFR's)



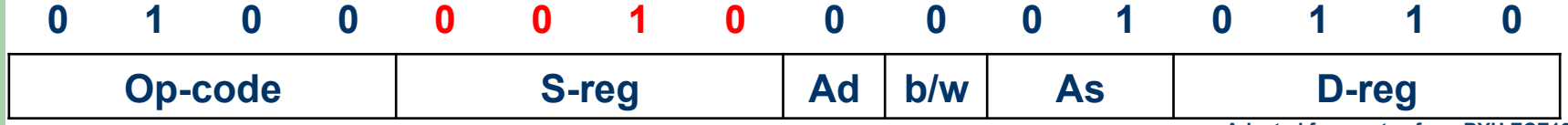
–Adapted from notes from BYU ECE124

Absolute Mode (&label)

- The address is formed directly from a constant (index) and specified by preceding a label with an ampersand (&)
  - **Example:** (mov.w x(SR), r6 where 0 is used for SR)

```
mov.w &Cnt,r6        ; move word
                     ; M(Cnt) to r6
```

- Same as indexed mode with the base register value of 0 (by using the status register SR as the base register)
- The absolute address is stored in the memory word following the instruction and requires an additional cycle
- Note: this is the preferred mode of addressing when referencing fixed locations in memory such as the special function registers (SFR's)



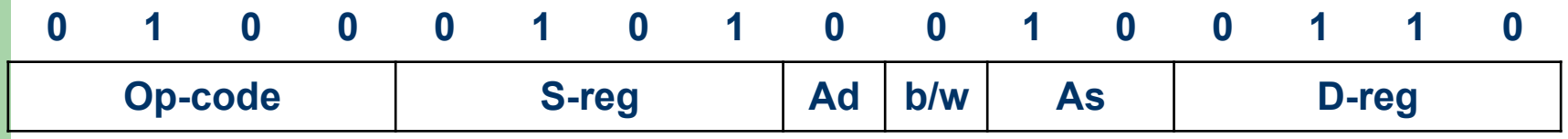
–Adapted from notes from BYU ECE124

Indirect Register Mode (@Rn)

- The address of the operand is formed from the contents of the specified register
  - **Example:**

```
mov.w @r5,r6      ; move word
                  ; M(r5) to r6
```

- Only available for source operands
- Same as indexed mode with index equal to 0, but does not require an additional instruction word
- The value of the indirect register is unchanged



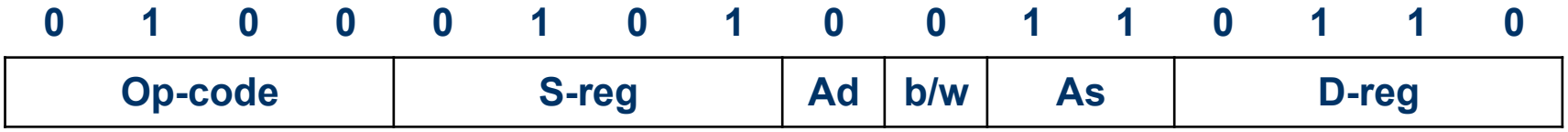
–Adapted from notes from BYU ECE124

Indirect Autoincrement Mode (@Rn+)

- The address of the operand is formed from the contents of the specified register and afterwards, the register is automatically increment by 1 if a byte is fetched or by 2 if a word is fetched
  - **Example:**

```
mov.w @r5+,r6      ; move word
                   ; M(r5) to r6
                   ; increment r5 by 2
```

- Only available for source operands.
- Usually called post-increment addressing.
- **Note: All operations on the first address are fully completed before the second address is evaluated**

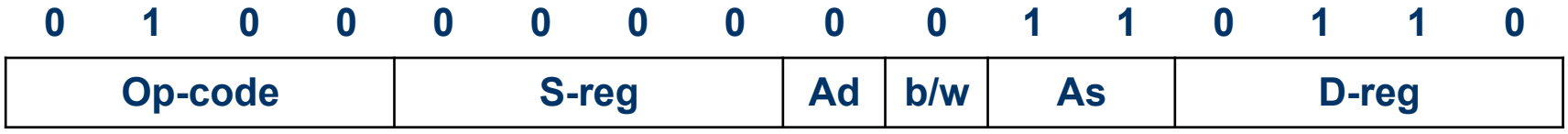


–Adapted from notes from BYU ECE124

Immediate Mode (#n)

- The operand is an immediate value
  - **Example** (mov.w @PC+, r6)

    mov.w #100,r6 ; 100 -> r6
- The immediate value is located in the memory word following the instruction
- Only available for source operands
- The immediate mode of addressing is a special case of auto-increment addressing that uses the program counter (PC) as the source register.
- The PC is automatically incremented after the instruction is fetched; hence points to the following word



–Adapted from notes from BYU ECE124

Constant Generators

- The following source register/addressing mode combinations result in a commonly used constant operand value
- Do not require an additional instruction word

Register	As	Constant	Remarks
R2	00	– – – – –	Register mode
R2	01	(0)	Absolute address 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

–Adapted from notes from BYU ECE124

Addressing Summary

ADDRESS MODE	S	D	SYNTAX	EXAMPLE	OPERATION
Register	●	●	MOV Rs,Rd	MOV R10,R11	R10 --> R11
Indexed	●	●	MOV X(Rn),Y(Rm)	MOV 2(R5),6(R6)	M(2+R5)--> M(6+R6)
Symbolic (PC relative)	●	●	MOV EDE,TONI		M(EDE) --> M(TONI)
Absolute	●	●	MOV &MEM,&TCDAT		M(MEM) --> M(TCDAT)
Indirect	●		MOV @Rn,Y(Rm)	MOV @R10,Tab(R6)	M(R10) --> M(Tab+R6)
Indirect autoincrement	●		MOV @Rn+,Rm	MOV @R10+,R11	M(R10) --> R11 R10 + 2--> R10
Immediate	●		MOV #X,TONI	MOV #45,TONI	#45 --> M(TONI)

NOTE: S = source D = destination

–Adapted from notes from BYU ECE124

Addressing Modes

As/Ad	Addressing Mode	Syntax	Description
00/0	Register mode	Rn	Register contents are operand
01/1	Indexed mode	X(Rn)	(Rn + X) points to the operand. X is stored in the next word.
01/1	Symbolic mode	ADDR	(PC + X) points to the operand. X is stored in the next word. Indexed mode X(PC) is used.
01/1	Absolute mode	&ADDR	The word following the instruction contains the absolute address. X is stored in the next word. Indexed mode X(SR) is used.
10/–	Indirect register mode	@Rn	Rn is used as a pointer to the operand.
11/–	Indirect autoincrement	@Rn+	Rn is used as a pointer to the operand. Rn is incremented afterwards by 1 for .B instructions and by 2 for .W instructions.
11/–	Immediate mode	#N	The word following the instruction contains the immediate constant N. Indirect autoincrement mode @PC+ is used.

–Adapted from notes from BYU ECE124

Format I: Double Operand

- 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

–Adapted from notes from BYU ECE124

Example: Double Operand

- Copy the contents of a register to another register
  - Assembly: **mov.w r5,r4**
  - Instruction code: **0x4504**

Op-code mov	*S-reg r5*	*Ad Register*	*b/w 16-bits*	*As Register*	*D-reg r4*
0 1 0 0	**0 1 0 1**	**0**	**0**	**0 0**	**0 1 0 0**

- One word instruction
- The instruction instructs the CPU to copy the 16-bit 2's complement number in register **r5** to register **r4**

Example: Double Operand

- Copy the contents of a register to a PC-relative memory address location
  - Assembly: **mov.w r5,TONI**
  - Instruction code: **0x4580**

Op-code mov	*S-reg r5*	*Ad Symbolic*	*b/w 16-bits*	*As Register*	*D-reg PC*
0 1 0 0	**0 1 0 1**	**1**	**0**	**0 0**	**0 0 0 0**
2's complement PC-relative destination index					

- Two word instruction
- The instruction instructs the CPU to copy the 16-bit 2's complement word in register **r5** to the memory location whose address is obtained by adding the **PC** to the memory word following the instruction

–Adapted from notes from BYU ECE124

Example: Double Operand

- Copy the contents of a PC-relative memory location to another PC-relative memory location
  - Assembly: **mov.b EDEN,TONI**
  - Instruction code: **0x40d0**

Op-code mov	S-reg PC	Ad Symbolic	b/w 8-bits	As Symbolic	D-reg PC
0 1 0 0	0 0 0 0	1	1	0 1	0 0 0 0
2's complement PC-relative source index					
2's complement PC-relative destination index					

- Three word instruction
- The CPU copies the 8-bit contents of EDEN (pointed to by **`source index + PC`**) to TONI (pointed to by **`destination index + PC`**)

–Adapted from notes from BYU ECE124

Format II: Single Operand

- 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 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`	@SP+→SR, @SP+→SP	Return from interrupt

–Adapted from notes from BYU ECE124

Example: Single Operand

- Logically shift the contents of register **r5** to the right through the status register carry
  - Assembly: **rrc.w r5**
  - Instruction code: **0x1005**

Op-code rrc	*b/w 16-bits*	*Ad Register*	D-reg *r5*
0 0 0 1 0 0 0 0 0	**0**	**0 0**	**0 1 0 1**

- One word instruction
- The CPU shifts the 16-bit register **r5** one bit to the right (divide by 2) – the carry bit prior to the instruction becomes the MSB of the result while the LSB shifted out replaces the carry bit in the status register

–Adapted from notes from BYU ECE124

Example: Single Operand

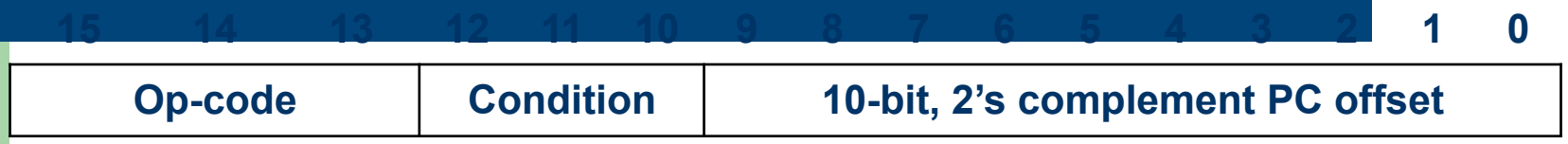
- Arithmetically shift the contents of absolute memory location **P2OUT** to the right through the SR carry
  - Assembly: **rra.b &P2OUT**
  - Instruction code: **0x1152**

Op-code *rra*	*b/w* *8-bits*	*Ad* *Indexed*	D-reg *r2*
0 0 0 1 0 0 0 1 0	1	0 1	0 0 1 0
Absolute memory address (P2OUT)			

- Two word instruction
- The CPU arithmetically shifts the 8-bit memory location **`P2OUT`** one bit to the right (divide by 2) – MSB prior to the instruction becomes the MSB of the result while the LSB shifted out replaces the carry bit in the SR

–Adapted from notes from BYU ECE124

Jump Instruction Format



- Jump instructions are used to direct program flow to another part of the program.
- The condition on which a jump occurs depends on the Condition field consisting of 3 bits:
  - 000: jump if not equal
  - 001: jump if equal
  - 010: jump if carry flag equal to zero
  - 011: jump if carry flag equal to one
  - 100: jump if negative (N = 1)
  - 101: jump if greater than or equal (N = V)
  - 110: jump if lower (N ≠ V)
  - 111: unconditional jump

–Adapted from notes from BYU ECE124

Jump Instruction Format

- Jump instructions are executed based on the current PC and the status register
- Conditional jumps are controlled by the status bits
- Status bits are not changed by a jump instruction
- The jump off-set is represented by the 10-bit, 2's complement value:

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

- Thus, the range of the jump is -511 to +512 words, (-1022 to 1024 bytes) from the current instruction
- Note: Use a BR instruction to jump to any address

–Adapted from notes from BYU ECE124

Example: Jump Format

- Continue execution at the label **main** if the carry bit is set
  - Assembly: **jc main**
  - Instruction code: **0x2fe4**

Op-code JC	*Condition Carry Set*	*10-Bit, 2's complement PC offset -28*
0 0 1	**0 1 1**	**1 1 1 1 1 0 0 1 0 0**

- One word instruction
- The CPU will add to the **PC** (R0) the value **-28 x 2** if the carry is set

–Adapted from notes from BYU ECE124

Emulated Instructions

- 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 easier, 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.

–Adapted from notes from BYU ECE124

Emulated Instructions

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	d s t + C → d s t (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.

–Adapted from notes from BYU ECE124

Emulated Instructions

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

–Adapted from notes from BYU ECE124

Emulated Instructions

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

–Adapted from notes from BYU ECE124

Example: Emulated Instructions

- Clear the contents of register R5:
  - `CLR R5`
  - Instruction code: 0x4305

Op-code mov	*S-reg r3*	*Ad Register*	*b/w 16-bits*	*As Register*	*D-reg r5*
0 1 0 0	0 0 1 1	0	0	0 0	0 1 0 1

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

Adapted from notes from BYU ECE124

Example: Emulated Instructions

- Increment the content of register R5:
  - `INC R5`
  - Instruction code: 0x5315

Op-code add	*S-reg r3*	*Ad Register*	*b/w 16-bits*	*As Indexed*	*D-reg r5*
0 1 0 1	0 0 1 1	0	0	0 1	0 1 0 1

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

–Adapted from notes from BYU ECE124

Example: Emulated Instructions

- Decrement the contents of register R5:
  - `DEC R5`
  - Instruction code: 0x8315

Op-code sub	S-reg r3	Ad Register	b/w 16-bits	As Indexed	D-reg r5
1 0 0 0	0 0 1 1	0	0	0 1	0 1 0 1

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

Adapted from notes from BYU ECE124

Example: Emulated Instructions

- Decrement by two the contents of register R5:
  - `DECD R5`
    - Instruction code: 0x8325

Op-code sub	S-reg r3	Ad Register	b/w 16-bits	As Indirect	D-reg r5
1 0 0 0	0 0 1 1	0	0	1 0	0 1 0 1

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

–Adapted from notes from BYU ECE124

Example: Emulated Instructions

- Do not carry out any operation:
  - **NOP**
    - Instruction code: 0x4303

Op-code mov	S-reg r3	Ad Register	b/w 16-bits	As Register	D-reg r5
0 1 0 0	0 0 1 1	0	0	0 0	0 0 1 1

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

Adapted from notes from BYU ECE124

Example: Emulated Instructions

- Add the carry flag to the register R5:
  - `ADC R5`
  - Instruction code: 0x6305

Op-code addc	S-reg r3	Ad Register	b/w 16-bits	As Register	D-reg r5
0 1 1 0	0 0 1 1	0	0	0 0	0 1 0 1

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

Assembly to Machine Code

–Memory Location

–Machine code instruction

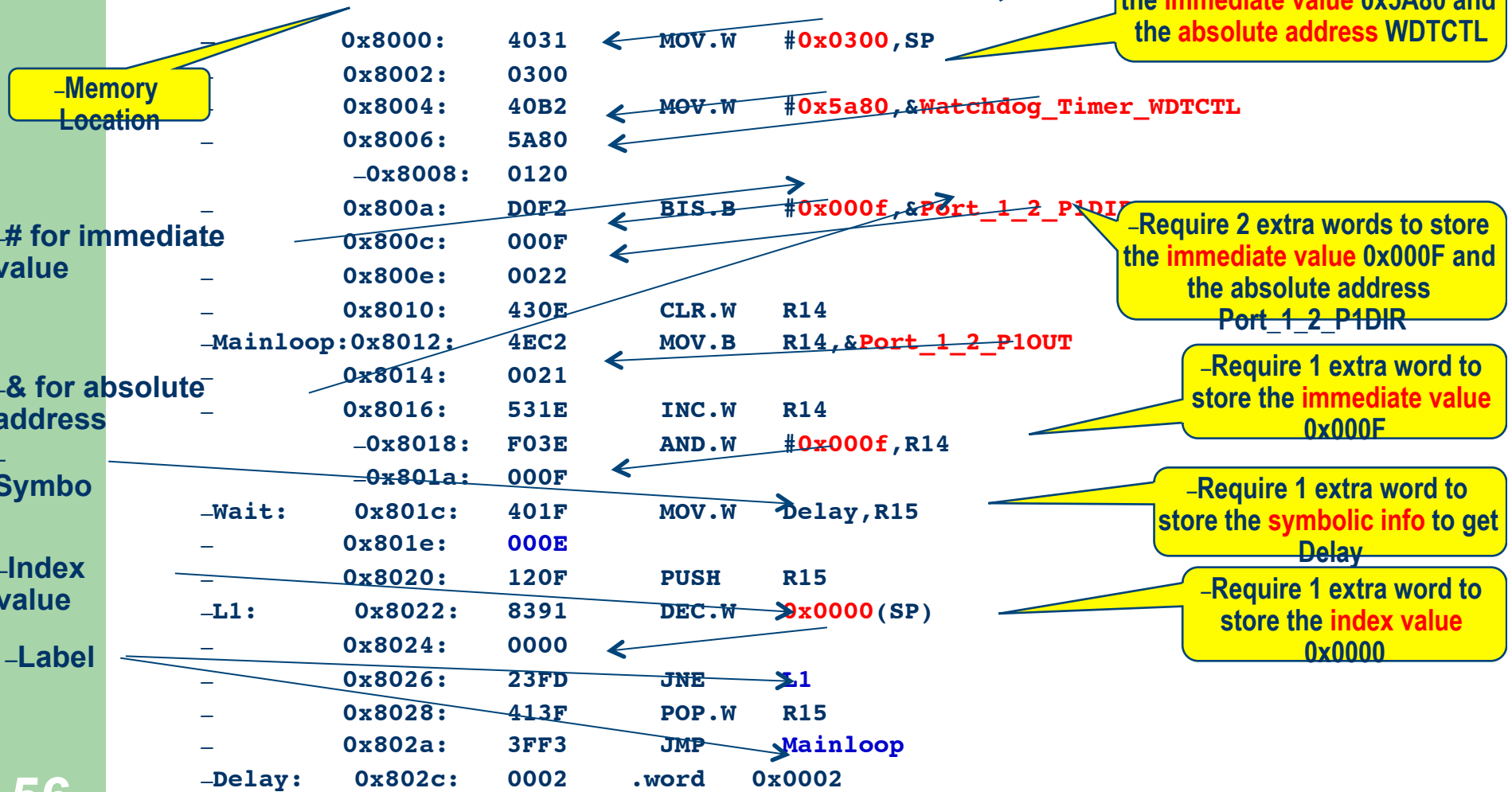
–Machine code information

–Assembly code

```
–0x8000:   4031       0300          MOV.W   #0x0300,SP
–0x8004:   40B2       5A80 0120     MOV.W   #0x5a80,&Watchdog_Timer_WDTCTL
–0x800a:   D0F2       000F 0022     BIS.B   #0x000f,&Port_1_2_P1DIR
–0x8010:   430E                     CLR.W   R14
–        Mainloop:
–0x8012:   4EC2       0021          MOV.B   R14,&Port_1_2_P1OUT
–0x8016:   531E                     INC.W   R14
–0x8018:   F03E       000F          AND.W   #0x000f,R14
–        Wait:
–0x801c:   401F       000E          MOV.W   Delay,R15
–0x8020:   120F                     PUSH    R15
–        L1:
–0x8022:   8391       0000          DEC.W   0x0000(SP)
–0x8026:   23FD                     JNE     (L1)
–0x8028:   413F                     POP.W   R15
–0x802a:   3FF3                     JMP     (Mainloop)
–        Delay:
–0x802c:   0002                     .word   0x0002
```


–Adapted from notes from BYU ECE124

Machine Code in the Memory



–Adapted from notes from BYU ECE124

Memory Location Offset

```
–          0x8000:    4031       MOV.W    #0x0300,SP
–          0x8002:    0300
–          0x8004:    40B2       MOV.W    #0x5a80,&Watchdog_Timer_WDTCTL
–          0x8006:    5A80
           –0x8008:   0120
–          0x800a:    D0F2       BIS.B    #0x000f,&Port_1_2_P1DIR
–          0x800c:    000F
–          0x800e:    0022
–          0x8010:    430E       CLR.W    R14
–Mainloop:0x8012:     4EC2       MOV.B    R14,&Port_1_2_P1OUT
–          0x8014:    0021
–          0x8016:    531E       INC.W    R14
           –0x8018:   F03E       AND.W    #0x000f,R14
           –0x801a:   000F
–Wait:     0x801c:    401F       MOV.W    Delay,R15
–          0x801e:    000E
0x000E
–          0x8020:    120F       PUSH     R15
–L1:       0x8022:    8391       DEC.W    0x0000(S
–          0x8024:    0000
–          0x8026:    23FD       JNE      L1
–          0x8028:    413F       POP.W    R15
–          0x802a:    3FF3       JMP      Mainloop
–Delay:    0x802c:    0002     .word    0x0002
```

–0x8028-0x8022=0x0006
jump -6 bytes or -3 words)
–JNE 0000 0011 1111 1101

–New PC value

–New PC value

(14) 0x802c-0x801e = 0x000E

–New PC value

–0x802c-0x8012=0x001a
jump -26 bytes or -13 words)
–JMP 0001 1111 1111 0011

–Adapted from notes from BYU ECE124

Machine Code in the Memory

```
          0x8000:                    MOV.W   #0x0300,SP
          0x8002:    0300
          0x8004:                    MOV.W   #0x5a80,&Watchdog_Timer_WDTCTL
          0x8006:    5A80
          0x8008:    0120
          0x800a:                    BIS.B   #0x000f,&Port_1_2_P1DIR
          0x800c:    000F
          0x800e:    0022
          0x8010:                    CLR.W   R14
Mainloop: 0x8012:                    MOV.B   R14,&Port_1_2_P1OUT
          0x8014:    0021
          0x8016:                    INC.W   R14
          0x8018:                    AND.W   #0x000f,R14
          0x801a:    000F
Wait:     0x801c:                    MOV.W   Delay,R15
          0x801e:    000E
          0x8020:                    PUSH    R15
L1:       0x8022:                    DEC.W   0x0000(SP)
          0x8024:    0000
          0x8026:                    JNE     L1
          0x8028:                    POP.W   R15
          0x802a:                    JMP     Mainloop
Delay:    0x802c:    0002            .word   0x0002
```


Adapted from notes from BYU ECE124

Machine Code in the Memory

```
–          0x8000:    4031  0100 0000 0011 0001          MOV.W    #0x0300,SP
–          0x8002:    0300  0000 0011 0000 0000
–          0x8004:    40B2  0100 0000 1011 0010          MOV.W
#0x5a80,&Watchdog_Timer_WDTCTL
–          0x8006:    5A80  0101 1010 1000 0000
           –0x8008:   0120  0000 0001 0010 0000
–          0x800a:    D0F2  1101 0000 1111 0010          BIS.B    #0x000f,&Port_1_2_P1DIR
–          0x800c:    000F  0000 0000 0000 1111
–          0x800e:    0022  0000 0000 0010 0010
–          0x8010:    430E  0100 0011 0000 1110          CLR.W    R14     ;(MOV.W #0X0000, R14)
–Mainloop:0x8012:     4EC2  0100 1110 1100 0010          MOV.B    R14, &Port_1_2_P1OUT
–          0x8014:    0021  0000 0000 0010 0001
–          0x8016:    531E  0101 0011 0001 1110          INC.W    R14     ;(ADD.W #0X01, R14)
           –0x8018:   F03E  1111 0000 0011 1110          AND.W    #0x000f,R14
           –0x801a:   000F  0000 0000 0000 1111
–Wait:     0x801c:    401F  0100 0000 0001 1111          MOV.W    Delay,R15
–          0x801e:    000E  0000 0000 0000 1110
–          0x8020:    120F  0001 0010 0000 1111          PUSH     R15
–L1:       0x8022:    8391  1000 0011 1001 0001          DEC.W    0(SP) ;(SUB.W #0X01, 0(SP)
–          0x8024:    0000  0000 0000 0000 0000
–          0x8026:    23FD  0010 0011 1111 1101          JNE      L1
–          0x8028:    413F  0100 0001 0011 1111          POP.W    R15    ;(MOV.W @SP+, R15)
–          0x802a:    3FF3  0011 1111 1111 0011          JMP      Mainloop
–Delay:    0x802c:    0002  0000 0000 0000 0010     .word    0x0002
```


–Adapted from notes from BYU ECE124

Practice:

- Disassemble the following MSP430 instructions:

Address	Data	
0x8010:	4031 0100 0000 0011 0001	mov.w #0x0600,r1
0x8012:	0600	
0x8014:	40B2 0100 0000 1011 0010	mov.w #0x5a1e,&0x0120
0x8016:	5A1E	
0x8018:	0120	
0x801a:	430E 0100 0011 0000 1110	mov.w #0,r14
0x801c:	535E 0101 0011 0101 1110	add.b #1,r14
0x801e:	F07E 1111 0000 0111 1110	and.b #0x0f,r14
0x8020:	000F	
0x8022:	1230 0001 0010 0011 0000	push #0x000e
0x8024:	000E	
0x8026:	8391 1000 0011 1001 0001	sub.w #1, 0(r1)
0x8028:	0000	
0x802a:	23FD 0010 0011 1111 1101	jne 0x8026 (0x802C-3x2)
0x802c:	413F 0100 0001 0011 1111	mov.w @r1+,r15 (pop.w r15
0x802e:	3FF6 0011 1111 1111 0110	jmp 0x801c (0x8030-2x10)

–Adapted from notes from BYU ECE124