M2 – Instruction Set Architecture

Module Outline

- Addressing modes. Instruction classes.
- MIPS-I ISA.
- High level languages, Assembly languages and object code.
- Translating and starting a program.
- Subroutine and subroutine call. Use of stack for handling subroutine call and return.

Module Outline

- Addressing modes. Instruction classes.
- MIPS-I ISA.
- High level languages, Assembly languages and object code.
- Translating and starting a program.
- Subroutine and subroutine call. Use of stack for handling subroutine call and return.

Instruction Set Architecture

- Instruction Set: A vocabulary of commands understood by an architecture

Instruction Set Architecture

- Instruction Set: A vocabulary of commands understood by an architecture
- ISA includes:
  - All available instructions (**Instruction Set**)
  - How are operands specified (**Addressing modes**)
  - What does each instruction look like (**Instruction format**)

Instruction Set Architecture

- Instruction Set: A vocabulary of commands understood by an architecture
- ISA includes:
  - All available instructions (**Instruction Set**)
  - How are operands specified (**Addressing modes**)
  - What does each instruction look like (**Instruction format**)
- ISA design goal:
  - Find a language that makes it **easy to build the hardware** and **compiler** while **maximizing performance** and **minimizing cost.**

Types of Instructions

- Arithmetic and Logic Instructions
  -
  -
  -
- Data Transfer Instructions
  -
  -
  -

Types of Instructions

- Arithmetic and Logic Instructions
  - add, subtract, multiply, divide, compare (int/fp)
  - or, and, not, xor
  - shift (left/right, arithmetic/logical), rotate
- Data Transfer Instructions
  -
  -
  -

Types of Instructions

- Arithmetic and Logic Instructions
  - add, subtract, multiply, divide, compare (int/fp)
  - or, and, not, xor
  - shift (left/right, arithmetic/logical), rotate
- Data Transfer Instructions
  - load (copy data to a register from memory)
  - store (copy data to a memory location from a register)
  - move (copy data from one register to another)

Types of Instructions

- Control transfer instructions
  -
- Other instructions
  -

Types of Instructions

- Control transfer instructions
  - jump, conditional branch, function call, return
- Other instructions
  -

Types of Instructions

- Control transfer instructions
  - jump, conditional branch, function call, return
- Other instructions
  - eg. halt

MIPS-I Instruction Set

- Introduced in 1985 with the MIPS R2000.

MIPS-I Instruction Set

- Introduced in 1985 with the MIPS R2000.
- 32 General Purpose Registers (R0, R1, ... R31). Other data registers: HI, LO.
  -

MIPS-I Instruction Set

- Introduced in 1985 with the MIPS R2000.
- 32 General Purpose Registers (R0, R1, ... R31). Other data registers: HI, LO.
  - R0 is hardwired to 0
  - R31 is used as an implicit operand in some instructions

MIPS-I Instruction Set

- 32 General Purpose Registers (R0, R1, ... R31). Other data registers: HI, LO.

Name	Register number	Usage	Preserved on call?
$zero	0	The constant value 0	n.a.
$v0–$v1	2–3	Values for results and expression evaluation	no
$a0–$a3	4–7	Arguments	no
$t0–$t7	8–15	Temporaries	no
$s0–$s7	16–23	Saved	yes
$t8–$t9	24–25	More temporaries	no
$gp	28	Global pointer	yes
$sp	29	Stack pointer	yes
$fp	30	Frame pointer	yes
$ra	31	Return address	yes

MIPS-I Instruction Set

- Special registers: Program Counter (PC), Instruction Register (IR).

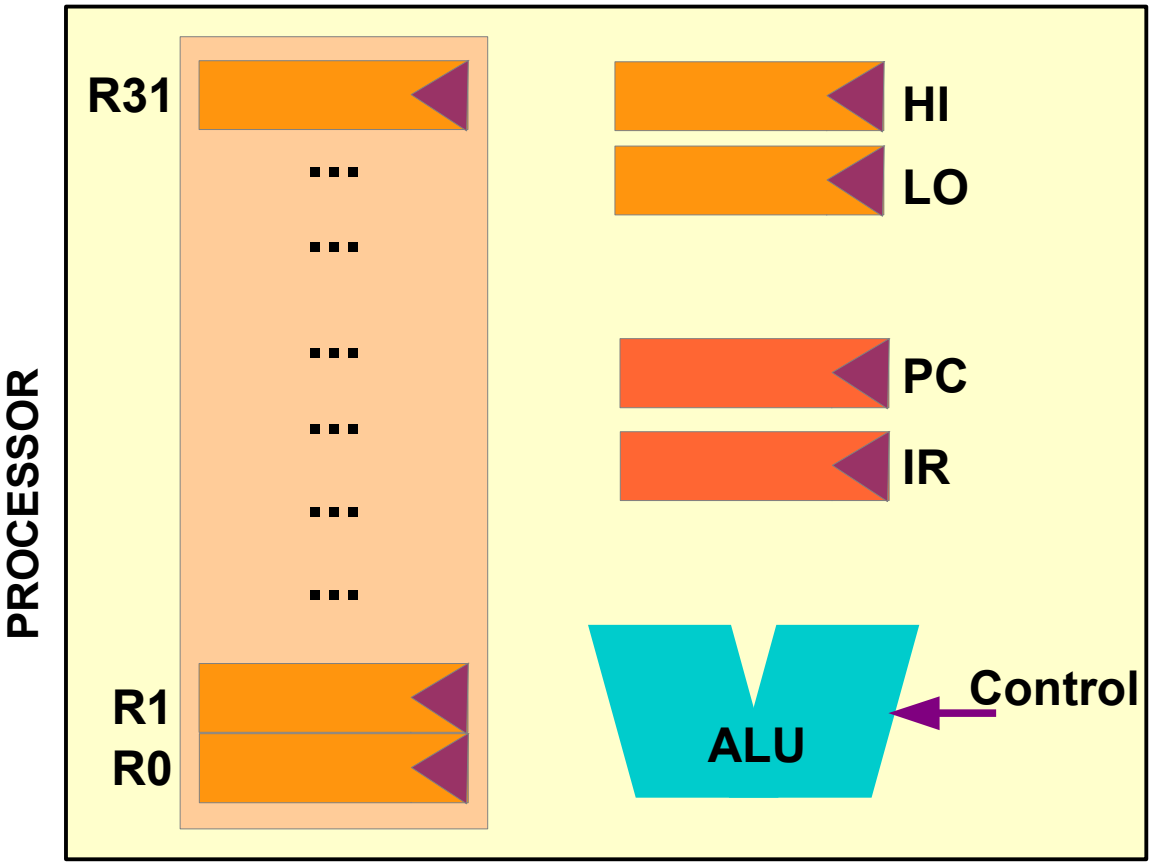
MIPS-I Instruction Set

- Special registers: Program Counter (PC), Instruction Register (IR).
- Instructions are 32 bits long

MIPS-I Instruction Set

- Special registers: Program Counter (PC), Instruction Register (IR).
- Instructions are 32 bits long
- Data Sizes:
  - byte(8 bits), halfword (2 bytes), word (4 bytes)
  - a character requires 1 byte of storage
  - an integer requires 1 word (4 bytes) of storage

MIPS Processor



- General purpose registers: R0 – R31
- Special Registers: PC, IR, Status Register.

MIPS-I Data Transfer Instructions

	Mnemonics	Example	Meaning
Load			
Store			
Move			

MIPS-I Data Transfer Instructions

	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	LW R2, 4(R3)	R2 ← Mem(R3 + 4)
Store			
Move			

- L: Load
- B: Byte (8b), H: Half Word (16b), W: Word (32b)
- U: Upper
- I: Immediate

MIPS-I Data Transfer Instructions

	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	LW R2, 4(R3)	R2 ← Mem(R3 + 4)
Store	SB, SH, SW	SB R2, -8(R4)	Mem(R4 - 8) ← R2
Move			

- L: Load
- S: Store

- B: Byte (8b), H: Half Word (16b), W: Word (32b)
- U: Upper
- I: Immediate

MIPS-I Data Transfer Instructions

	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	LW R2, 4(R3)	R2 ← Mem(R3 + 4)
Store	SB, SH, SW	SB R2, -8(R4)	Mem(R4 - 8) ← R2
Move	MFHI, MFLO, MTHI, MTLO	MFHI R1	R2 ← HI

- L: Load
- S: Store
- M: Move from/to HI/LO
- B: Byte (8b), H: Half Word (16b), W: Word (32b)
- U: Upper
- I: Immediate

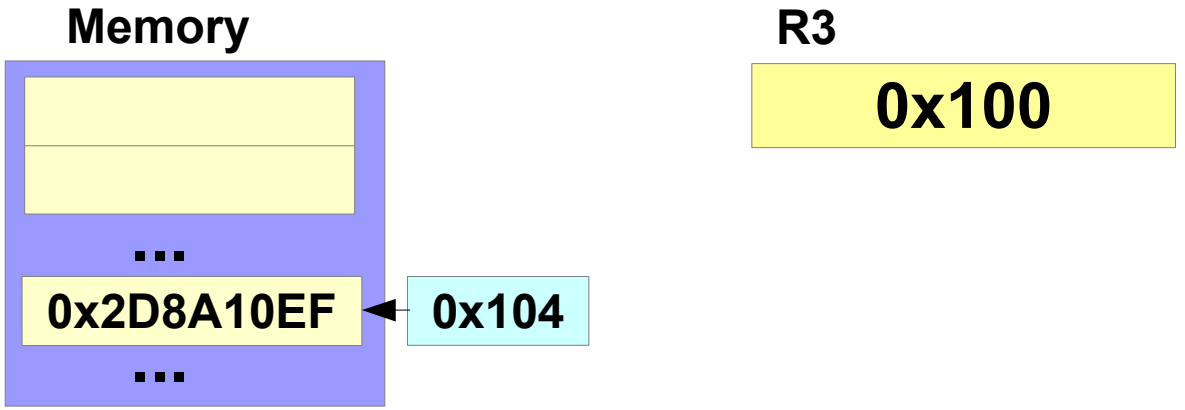
MIPS-I Data Transfer Instructions

	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	LB R2, 4(R3) LH R2, 4(R3) LW R2, 4(R3) LBU R2, 4(R3) LUI R2, 4(R3)	R2 ← Mem(R3 + 4)

- L: Load
- B: Byte (8b), H: Half Word (16b), W: Word (32b)
- U: Unsigned
- I: Immediate

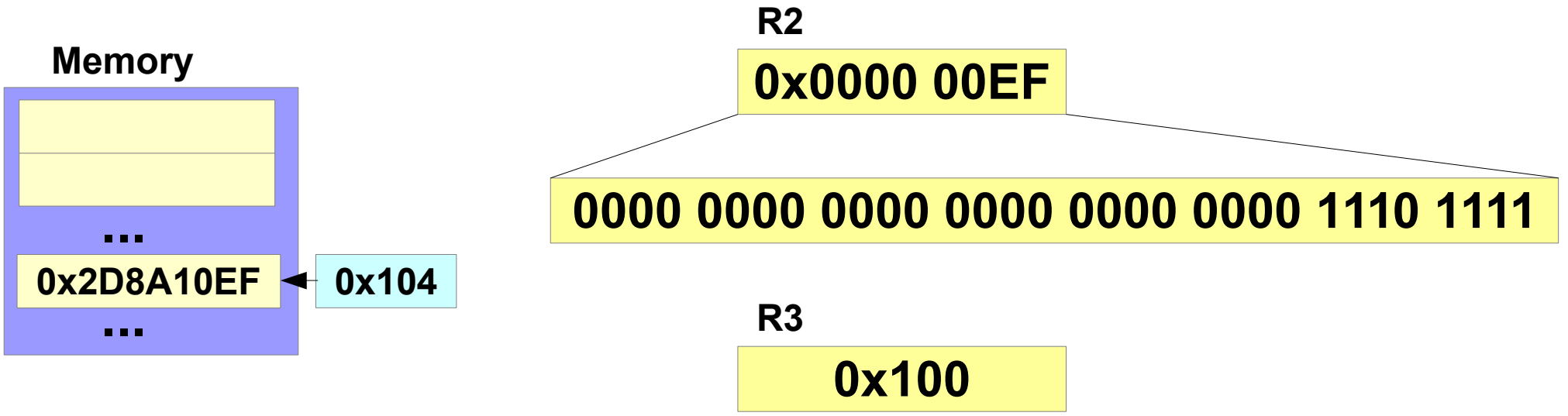
MIPS-I Data Transfer Instructions

	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	**LB R2, 4(R3)** LH R2, 4(R3) LW R2, 4(R3) LBU R2, 4(R3) LUI R2, 4(R3)	$R2_{7\text{-}0} \leftarrow Mem(R3 + 4)_{7\text{-}0}$



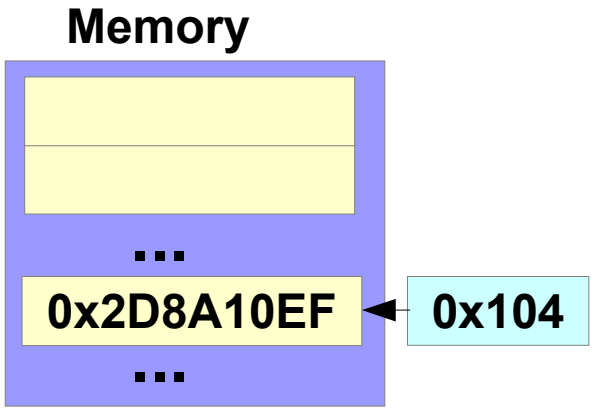
MIPS-I Data Transfer Instructions

	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	**LB R2, 4(R3)** LH R2, 4(R3) LW R2, 4(R3) LBU R2, 4(R3) LUI R2, 4(R3)	$R2_{7\text{-}0} \leftarrow Mem(R3 + 4)_{7\text{-}0}$



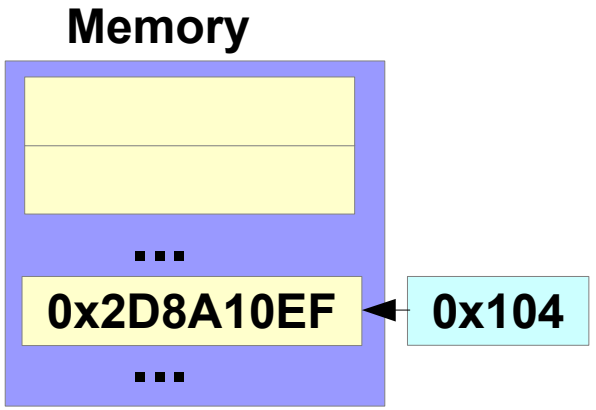
MIPS-I Data Transfer Instructions

	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	**LB R2, 4(R3)** LH R2, 4(R3) LW R2, 4(R3) **LBU R2, 4(R3)** LUI R2, 4(R3)	



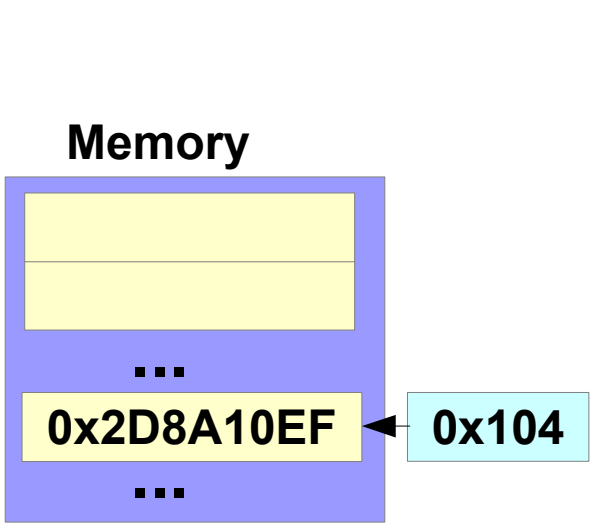
MIPS-I Data Transfer Instructions

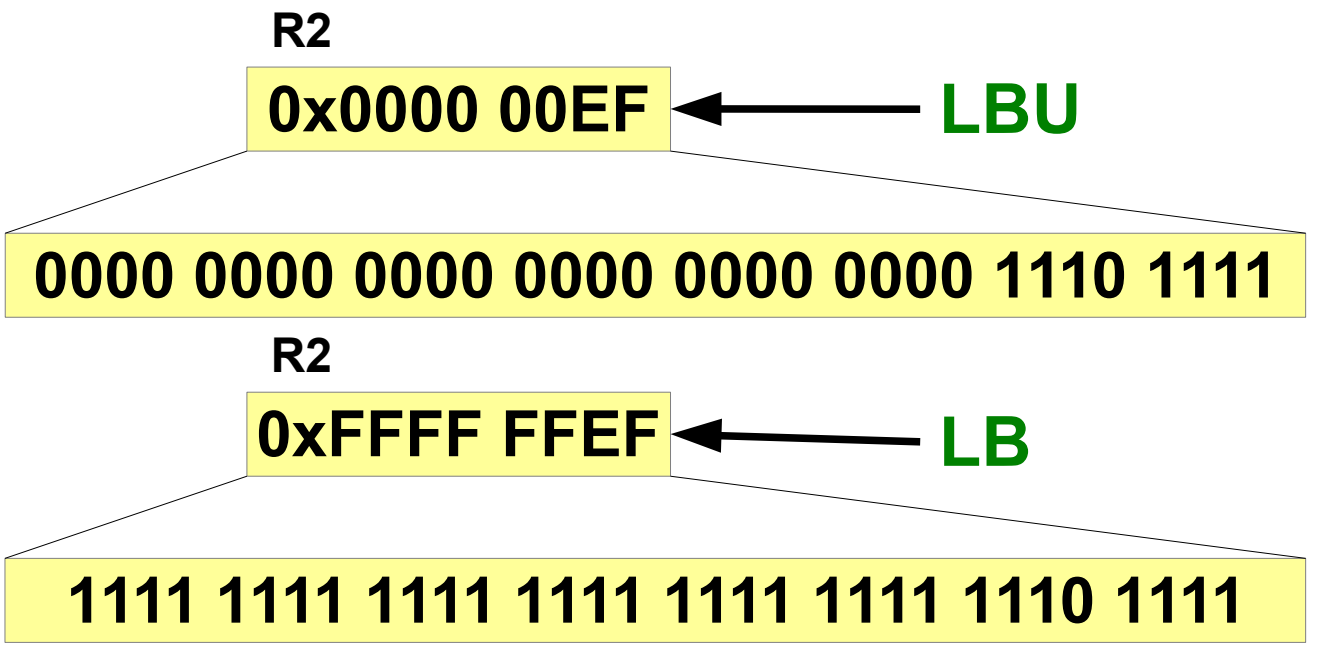
	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	**LB R2, 4(R3)**	$R2_{7-0} \leftarrow Mem(R3 + 4)_{7-0}$
		LH R2, 4(R3)	$R2_{31-8} \leftarrow$ Sign Extension
		LW R2, 4(R3)	
		LBU R2, 4(R3)	$R2_{7-0} \leftarrow Mem(R3 + 4)_{7-0}$
		LUI R2, 4(R3)	$R2_{31-8} \leftarrow$ Zero Extension



MIPS-I Data Transfer Instructions

	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	**LB R2, 4(R3)**	$R2_{7-0} \leftarrow Mem(R3 + 4)_{7-0}$
		LH R2, 4(R3)	$R2_{31-8} \leftarrow$ Sign Extension
		LW R2, 4(R3)	
		LBU R2, 4(R3)	$R2_{7-0} \leftarrow Mem(R3 + 4)_{7-0}$
		LUI R2, 4(R3)	$R2_{31-8} \leftarrow$ Zero Extension



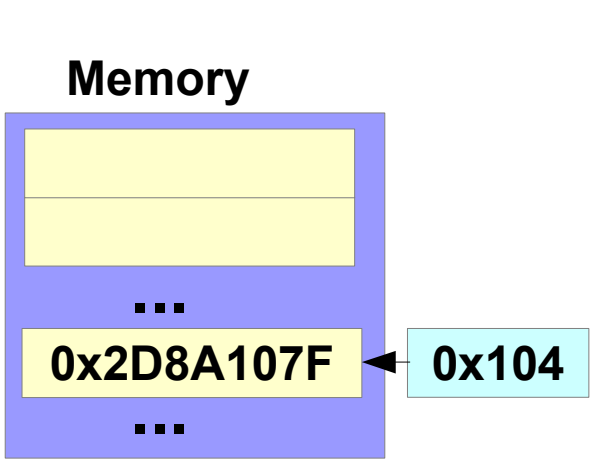


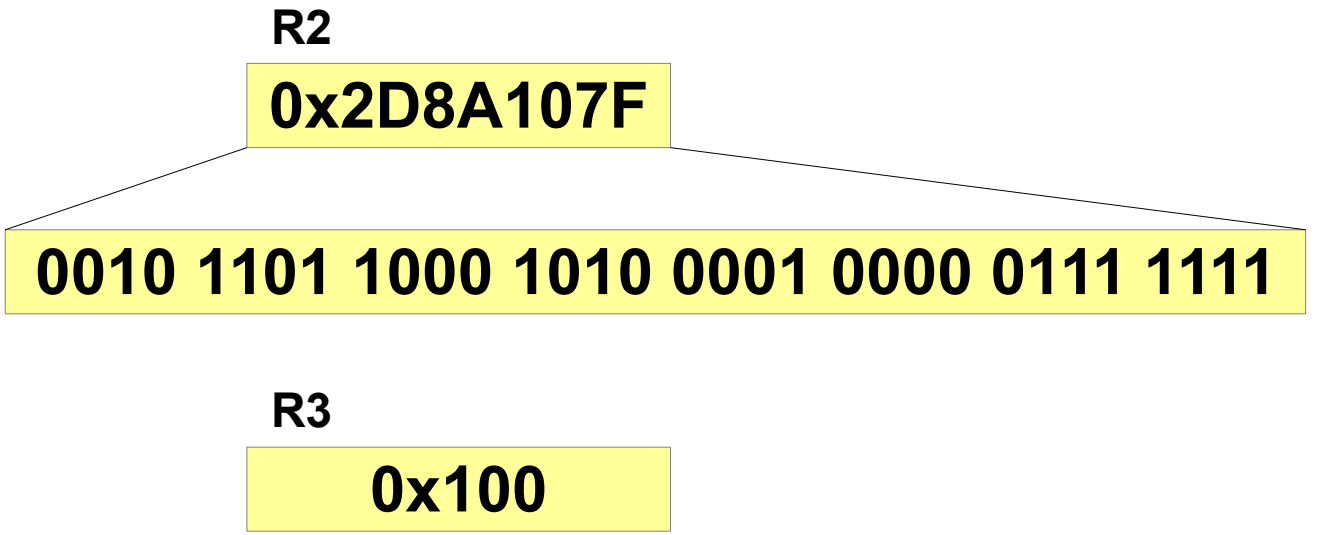
MIPS-I Data Transfer Instructions

	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	LB R2, 4(R3) LH R2, 4(R3) **LW R2, 4(R3)** LBU R2, 4(R3) LUI R2, 4(R3)	R2 ← Mem(R3 + 4)

MIPS-I Data Transfer Instructions

	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	LB R2, 4(R3) LH R2, 4(R3) **LW R2, 4(R3)** LBU R2, 4(R3) LUI R2, 4(R3)	R2 ← Mem(R3 + 4)



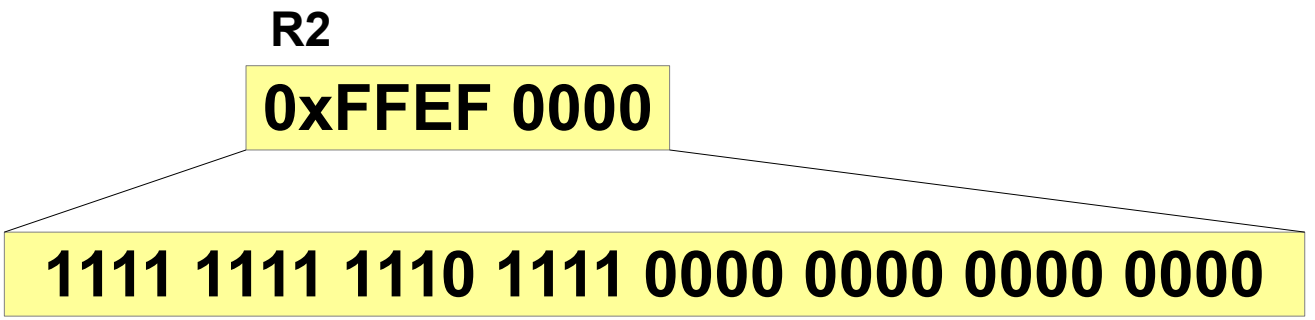


MIPS-I Data Transfer Instructions

	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	LB R2, 4(R3) LH R2, 4(R3) LW R2, 4(R3) LBU R2, 4(R3) **LUI R2, -17**	

MIPS-I Data Transfer Instructions

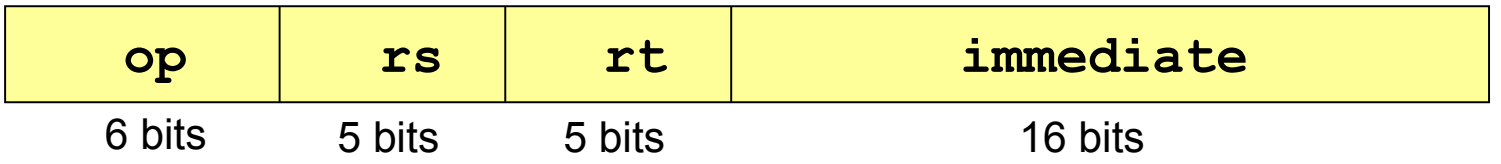
	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	LB R2, 4(R3) LH R2, 4(R3) LW R2, 4(R3) LBU R2, 4(R3) **LUI R2, -17**	$R2_{31\text{-}16} \leftarrow$ -17 $R2_{15\text{-}0} \leftarrow$ 00...0



MIPS-I Data Transfer Instructions

	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	LB R2, 4(R3) LH R2, 4(R3) LW R2, 4(R3) LBU R2, 4(R3) LUI R2, -17	

I Format



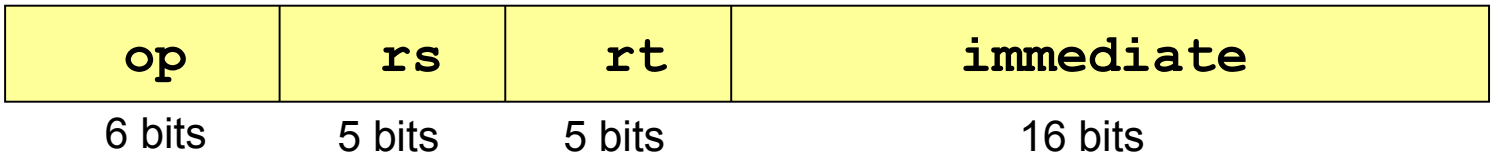
OP rt, rs, IMM

Encoding Example

- From the MIPS ISA Manual

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Load Upper Imm.	`lui`	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	`lw`	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}



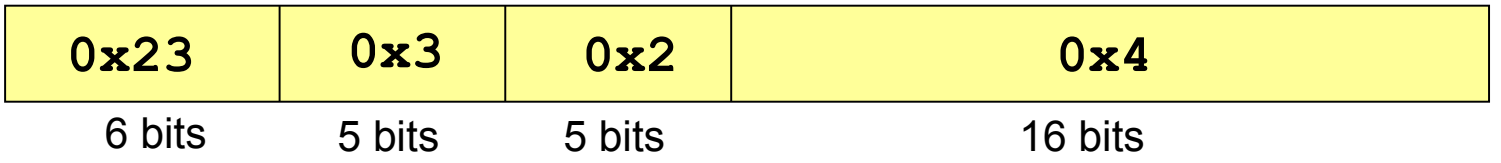
lw R2, 4(R3)

Encoding Example

- From the MIPS ISA Manual

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Load Upper Imm.	`lui`	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	`lw`	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}



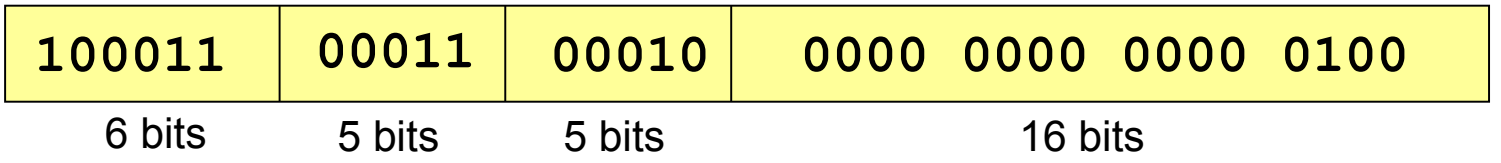
lw R2, 4(R3)

Encoding Example

- From the MIPS ISA Manual

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Load Upper Imm.	`lui`	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	`lw`	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}



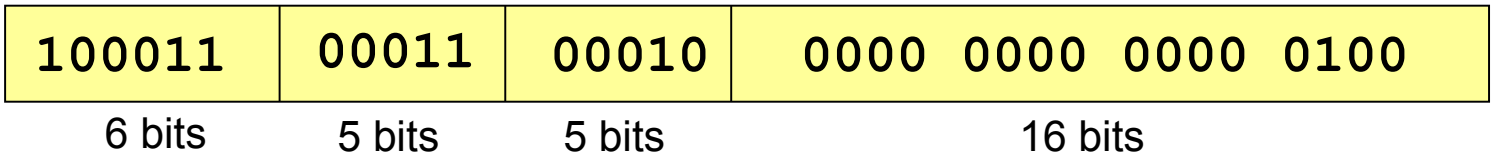
lw R2, 4(R3)

Encoding Example

- From the MIPS ISA Manual

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Load Upper Imm.	`lui`	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	`lw`	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}



lw R2, 4(R3)

0x8C620004

MIPS-I Data Transfer Instructions

	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	LW R2, 4(R3)	R2 ← Mem(R3 + 4)
Store	**SB, SH, SW**	**SB R2, -8(R4)**	**Mem(R4 - 8) ← R2**
Move	MFHI, MFLO, MTHI, MTLO	MFHI R1	R2 ← HI

- L: Load
- S: Store
- M: Move from/to HI/LO

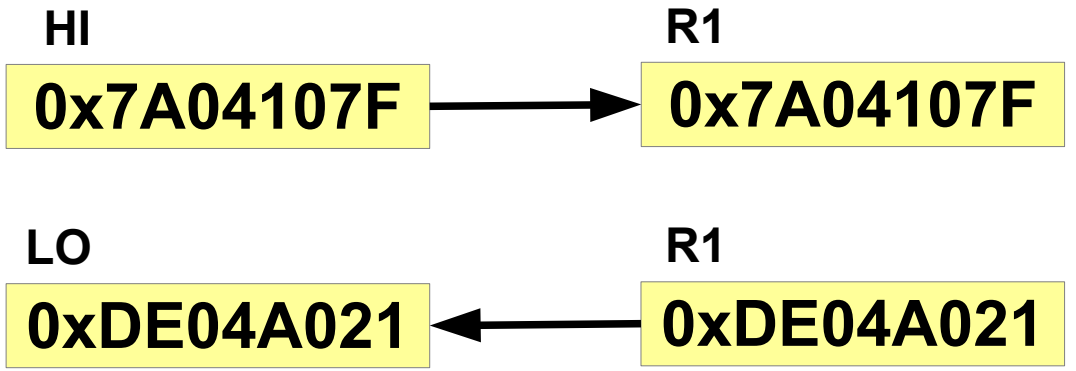
- B: Byte (8b), H: Half Word (16b), W: Word (32b)
- U: Upper
- I: Immediate

MIPS-I Data Transfer Instructions

	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	LW R2, 4(R3)	R2 ← Mem(R3 + 4)
Store	SB, SH, SW	SB R2, -8(R4)	Mem(R4 - 8) ← R2
Move	**MFHI, MFLO, MTHI, MTLO**	**MFHI R1** **MTLO R1**	**R1 ← HI** **LO ← R1**

MIPS-I Data Transfer Instructions

	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	LW R2, 4(R3)	R2 ← Mem(R3 + 4)
Store	SB, SH, SW	SB R2, -8(R4)	Mem(R4 - 8) ← R2
Move	**MFHI, MFLO, MTHI, MTLO**	**MFHI R1** **MTLO R1**	**R1 ← HI** **LO ← R1**



MIPS-I Arithmetic Instructions

	Mnemonics	Example	Meaning
Add, Subtract	ADD, ADDU, ADDI, ADDIU, SUB, SUBU		
Multiply, Divide	MULT, DIV, MULTU, DIVU		
Logical	AND, ANDI, OR, ORI, XOR, XORI, NOR		

MIPS-I Arithmetic Instructions

	Mnemonics	Example	Meaning
Add, Subtract	ADD, ADDU, ADDI, ADDIU, SUB, SUBU	**ADD R1, R2, R3** **ADDU R1, R2, R3** ADDI R1, R2, 6	R1 ← R2 + R3 R1 ← R2 + R3 R1 ← R2 + 6

MIPS-I Arithmetic Instructions

	Mnemonics	Example	Meaning
Add, Subtract	ADD, ADDU, ADDI, ADDIU, SUB, SUBU	**ADD R1, R2, R3** **ADDU R1, R2, R3** ADDI R1, R2, 6	R1 ← R2 + R3 R1 ← R2 + R3 R1 ← R2 + 6

- ADDU ignores overflow.
- Overflow: Result of an arithmetic operation is too large to be represented within the available bits.

Arithmetic Overflow

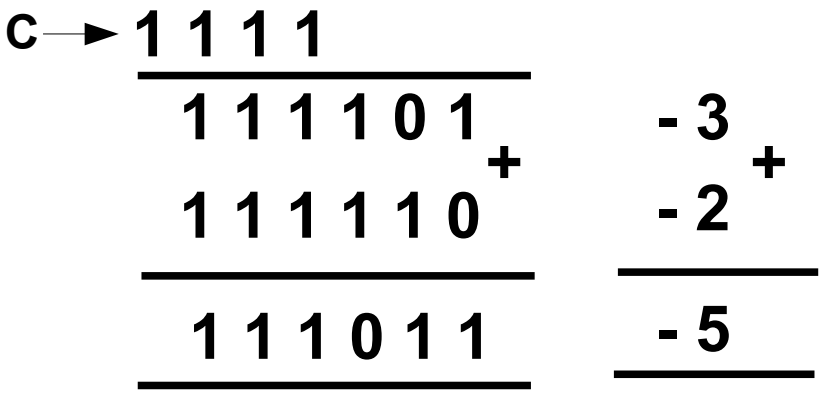
```
  1 1 1 1 0 1
+ 1 1 1 1 1 0
-------------

-------------
```

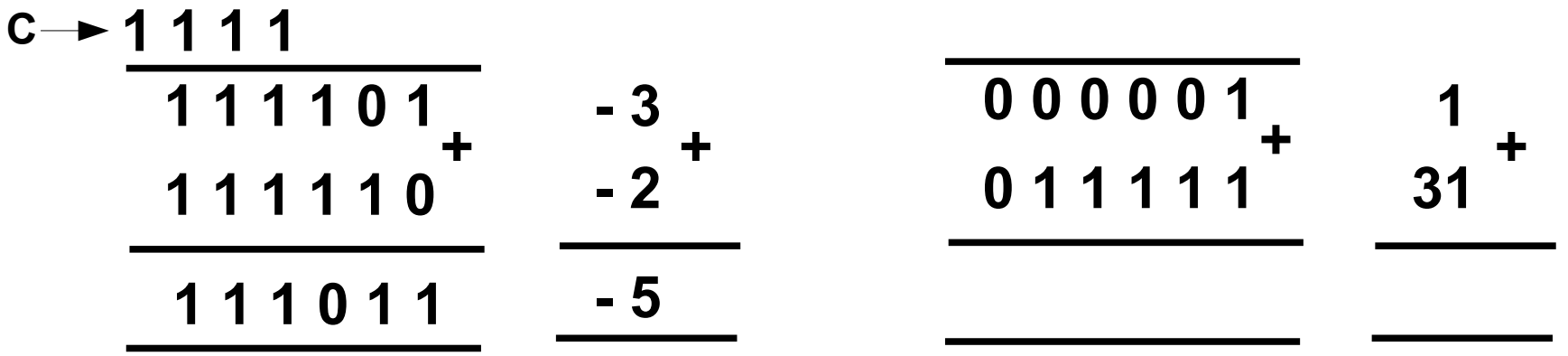
Arithmetic Overflow

1 1 1 1 0 1	- 3
+ 1 1 1 1 1 0	+ - 2

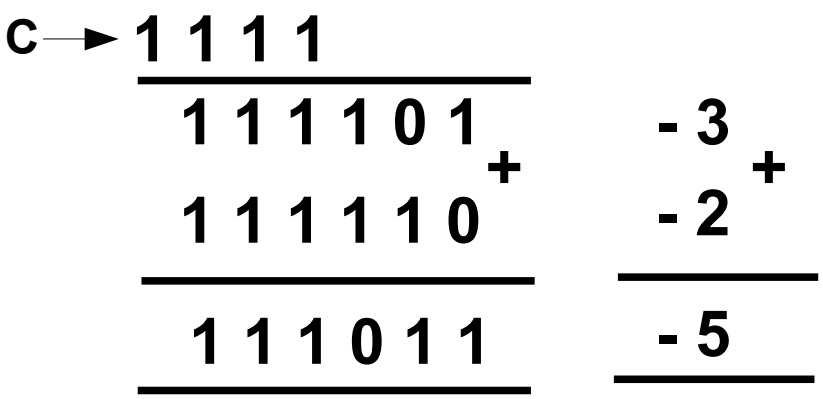
Arithmetic Overflow

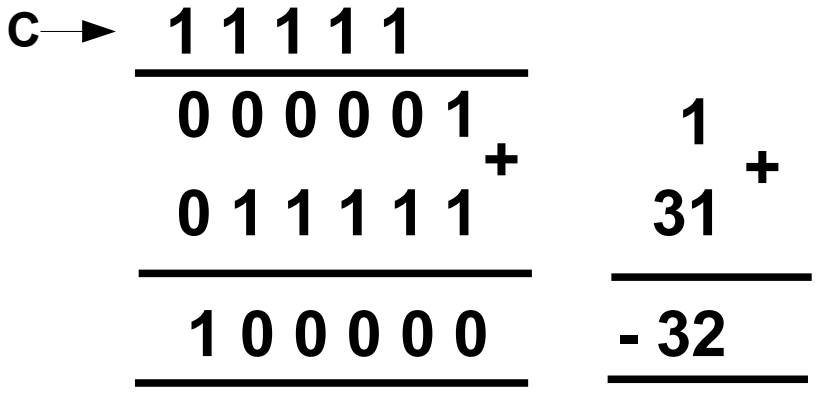


Arithmetic Overflow



Arithmetic Overflow





Arithmetic Overflow

```
c→ 1 1 1 1
   1 1 1 1 0 1      - 3
+  1 1 1 1 1 0   +  - 2
   ---------------  -----
   1 1 1 0 1 1      - 5
   ---------------  -----
```

```
c→ 1 1 1 1 1
   0 0 0 0 0 1        1
+  0 1 1 1 1 1   +   31
   ---------------  -----
   1 0 0 0 0 0      - 32
   ---------------  -----
```

```
   1 0 0 0 0 0     -32
+  1 1 1 1 1 1  +   -1
   ---------------  -----
   ---------------  -----
```

Arithmetic Overflow

```
c→ 1 1 1 1
   1 1 1 1 0 1      - 3
+  1 1 1 1 1 0   +  - 2
   ---------------  -----
   1 1 1 0 1 1      - 5
   ---------------  -----
```

```
c→ 1 1 1 1 1
   0 0 0 0 0 1        1
+  0 1 1 1 1 1   +   31
   ---------------  -----
   1 0 0 0 0 0      - 32
   ---------------  -----
```

```
c→ 1
   1 0 0 0 0 0     -32
+  1 1 1 1 1 1   +  -1
   ---------------  -----
   0 1 1 1 1 1      31
   ---------------  -----
```

Arithmetic Overflow

```
c→ 1 1 1 1
   ---------
   1 1 1 1 0 1     - 3
+  1 1 1 1 1 0  +  - 2
   ---------     ------
   1 1 1 0 1 1     - 5
   ---------     ------
```

```
c→  1 1 1 1 1
    -----------
    0 0 0 0 0 1      1
+   0 1 1 1 1 1  +  31
    -----------     ----
    1 0 0 0 0 0     -32
    -----------     ----
```

```
c→ 1
   ---------
   1 0 0 0 0 0    -32
+  1 1 1 1 1 1  +  -1
   ---------     ----
   0 1 1 1 1 1    31
   ---------     ----
```

- MSB0 = 0 && MSB1 = 0 && MSBofResult = 1
- MSB0 = 1 && MSB1 = 1 && MSBofResult = 0

MIPS-I Arithmetic Instructions

	Mnemonics	Example	Meaning

Add, Subtract	ADD, ADDU, ADDI, ADDIU, SUB, SUBU	ADD R1, R2, R3
		ADDU R1, R2, R3
		ADDI R1, R2, 6
			R1 ← R2 + R3
			R1 ← R2 + R3
			R1 ← R2 + 6

MIPS-I Arithmetic Instructions

	Mnemonics	Example	Meaning

Add, Subtract	ADD, ADDU, ADDI, ADDIU, SUB, SUBU	ADD R1, R2, R3
		ADDU R1, R2, R3
		ADDI R1, R2, 6
			R1 ← R2 + R3
			R1 ← R2 + R3
			R1 ← R2 + 6

- ADDI uses a 16 bit sign extended immediate operand

MIPS-I Arithmetic Instructions

	Mnemonics	Example	Meaning

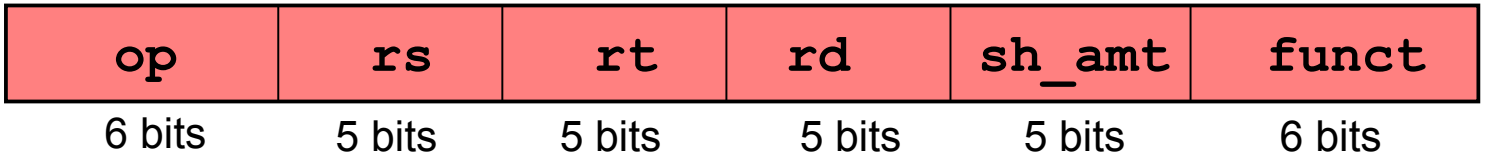
Add, Subtract	ADD, ADDU, ADDI, ADDIU, SUB, SUBU	ADD R1, R2, R3
		ADDU R1, R2, R3
		ADDI R1, R2, 6
			R1 ← R2 + R3
			R1 ← R2 + R3
			R1 ← R2 + 6

- Task: Move a 32b value into register R3.

MIPS-I Arithmetic Instructions

	Mnemonics	Example	Meaning
Add, Subtract	ADD, ADDU, ADDI, ADDIU, SUB, SUBU	**ADD R1, R2, R3** **ADDU R1, R2, R3** ADDI R1, R2, 6	R1 ← R2 + R3 R1 ← R2 + R3

R Format

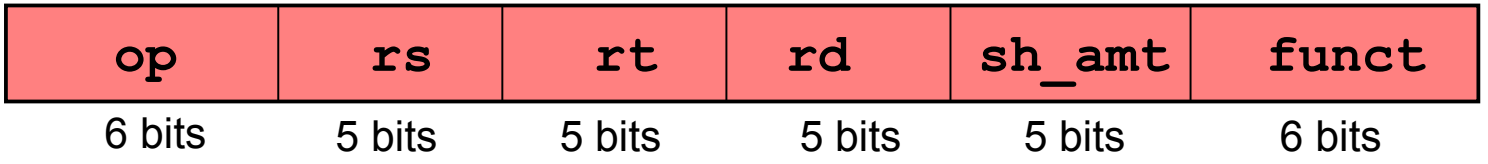


OP rd, rs, rt

MIPS-I Arithmetic Instructions

	Mnemonics	Example	Meaning
Add, Subtract	ADD, ADDU, ADDI, ADDIU, SUB, SUBU	**ADD R1, R2, R3** **ADDU R1, R2, R3** ADDI R1, R2, 6	R1 ← R2 + R3 R1 ← R2 + R3

R Format



OP rd, rs, rt

op: Opcode (class of instruction). Eg. ALU
funct: Which subunit of the ALU to activate?
ADD – **op/funct = 0x0/0x20**

sh_amt: Shift Amount. For Shift Instructions – **SLL, SRL**.

MIPS-I Arithmetic Instructions

	Mnemonics	Example	Meaning
Add, Subtract	ADD, ADDU, ADDI, ADDIU, SUB, SUBU	ADD R1, R2, R3 ADDI R1, R2, 6	R1 ← R2 + R3 R1 ← R2 + 6
Multiply, Divide	MULT, DIV, MULTU, DIVU	**MULT R1, R2**	**LO ← lsw (R1 * R2)** **HI ← msw (R1 * R2)**

MIPS-I Arithmetic Instructions

	Mnemonics	Example	Meaning
Add, Subtract	ADD, ADDU, ADDI, ADDIU, SUB, SUBU	ADD R1, R2, R3 ADDI R1, R2, 6	R1 ← R2 + R3 R1 ← R2 + 6
Multiply, Divide	MULT, DIV, MULTU, DIVU	**MULT R1, R2**	**LO ← lsw (R1 * R2)** **HI ← msw (R1 * R2)**

- Product of two 32-bit numbers is a 64-bit quantity. HI and LO contain the MSW and LSW of the Product

MIPS-I Arithmetic Instructions

	Mnemonics	Example	Meaning
Add, Subtract	ADD, ADDU, ADDI, ADDIU, SUB, SUBU	ADD R1, R2, R3 ADDI R1, R2, 6	R1 ← R2 + R3 R1 ← R2 + 6
Multiply, Divide	MULT, **DIV**, MULTU, DIVU	MULT R1, R2	LO ← lsw (R1 * R2) HI ← msw (R1 * R2)

- Divide Operation
  - Quotient in LO, Remainder in HI

MIPS-I Arithmetic Instructions

	Mnemonics	Example	Meaning
Add, Subtract	ADD, ADDU, ADDI, ADDIU, SUB, SUBU	ADD R1, R2, R3 ADDI R1, R2, 6	R1 ← R2 + R3 R1 ← R2 + 6
Multiply, Divide	MULT, DIV, MULTU, DIVU	MULT R1, R2	LO ← lsw (R1 * R2) HI ← msw (R1 * R2)
Logical	AND, ANDI, OR, ORI, XOR, XORI, NOR	OR R1, R2, 0xF	R1 ← R2 | SE(0xF)

MIPS-I Control Transfer Instructions

	Mnemonics	Example	Meaning
Conditional Branch	BEQ, BNE		
Jump	J, JR		
Jump and Link	JAL, JALR		
System Call	SYSCALL		

MIPS-I Control Transfer Instructions

	Mnemonics	Example	Meaning
Conditional Branch	BEQ, BNE	BEQ R2, R3, -16	If R2 == R3; PC ← PC + 4 -16
Jump	J, JR		
Jump and Link	JAL, JALR		
System Call	SYSCALL		

MIPS-I Control Transfer Instructions

	Mnemonics	Example	Meaning
Conditional Branch	BEQ, BNE	BEQ R2, R3, -16	If R2 == R3; PC ← PC + 4 -16
Jump	J, JR	J $target_{26}$	PC ← $PC_{31\text{-}28}$ || $target_{26}$ || 00
Jump and Link	JAL, JALR		
System Call	SYSCALL		

MIPS-I Control Transfer Instructions

	Mnemonics	Example	Meaning
Conditional Branch	BEQ, BNE	BEQ R2, R3, -16	If R2 == R3; PC ← PC + 4 -16
Jump	J, JR	J $target_{26}$	PC ← $PC_{31\text{-}28}$ || $target_{26}$ || 00
Jump and Link	JAL, JALR	JALR R2	R31 ← PC + 4 PC ← R2
System Call	SYSCALL		

MIPS-I Control Transfer Instructions

	Mnemonics	Example	Meaning
Conditional Branch	BEQ, BNE	BEQ R2, R3, -16	If R2 == R3; PC ← PC + 4 -16
Jump	J, JR	J $target_{26}$	PC ← $PC_{31\text{-}28}$ || $target_{26}$ || 00
Jump and Link	JAL, JALR	JALR R2	R31 ← PC + 4 PC ← R2
System Call	SYSCALL	SYSCALL	

MIPS-I Control Transfer Instructions

	Mnemonics	Example	Meaning
Conditional Branch	BEQ, BNE BGEZ, BLEZ BLTZ, BGTZ	BEQ R2, R3, -16	If R2 == R3; PC ← PC + 4 -16

MIPS-I Control Transfer Instructions

	Mnemonics	Example	Meaning
Conditional Branch	BEQ, BNE BGEZ, BLEZ BLTZ, BGTZ	BEQ R2, R3, -16	If R2 == R3; PC ← PC + 4 -16

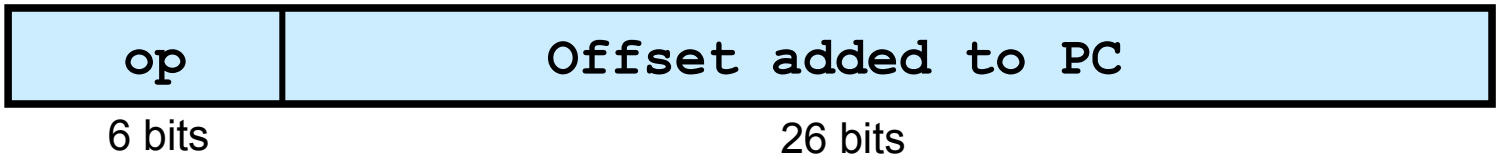
MIPS-I Control Transfer Instructions

	Mnemonics	Example	Meaning
Conditional Branch	BEQ, BNE	BEQ R2, R3, -16	If R2 == R3; PC ← PC + 4 -16
Jump	J, JR	J $target_{26}$ **J 0x475**	PC ← $PC_{31\text{-}28}$ || $target_{26}$ || 00

MIPS-I Control Transfer Instructions

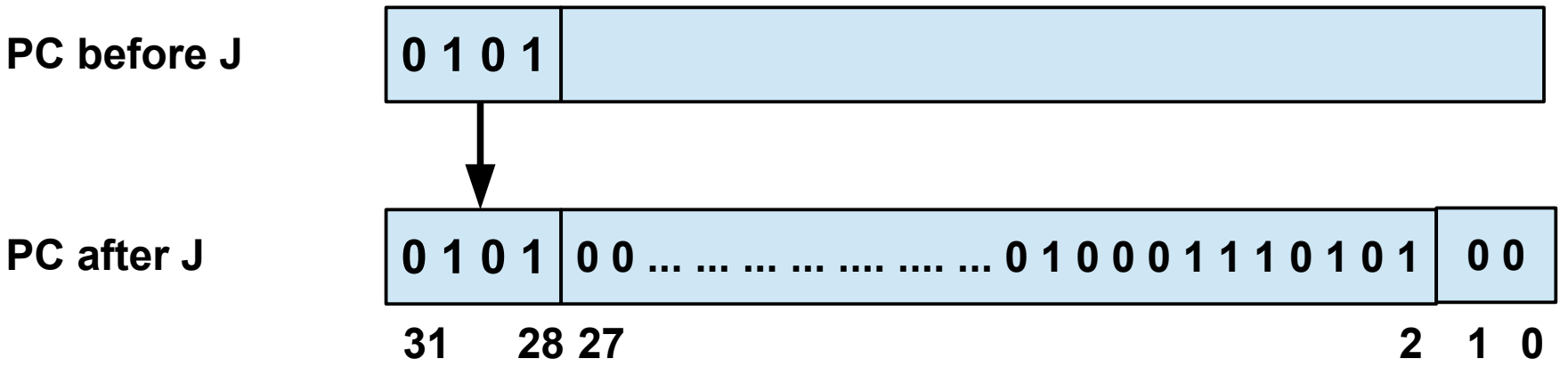
	Mnemonics	Example	Meaning
Conditional Branch	BEQ, BNE	BEQ R2, R3, -16	If R2 == R3; PC ← PC + 4 -16
Jump	J, JR	J $target_{26}$ **J 0x475**	PC ← $PC_{31\text{-}28}$ || $target_{26}$ || 00

J Format



MIPS-I Control Transfer Instructions

	Mnemonics	Example	Meaning
Conditional Branch	BEQ, BNE	BEQ R2, R3, -16	If R2 == R3; PC ← PC + 4 -16
Jump	J, JR	J $target_{26}$ **J 0x475**	PC ← $PC_{31\text{-}28}$ || $target_{26}$ || 00



MIPS-I Control Transfer Instructions

	Mnemonics	Example	Meaning
Conditional Branch	BEQ, BNE	BEQ R2, R3, -16	If R2 == R3; PC ← PC + 4 -16
Jump	J, JR	J $target_{26}$ J 0x475 **JR R2**	PC ← $PC_{31\text{-}28}$ || $target_{26}$ || 00 **PC ← R2**

MIPS-I Control Transfer Instructions

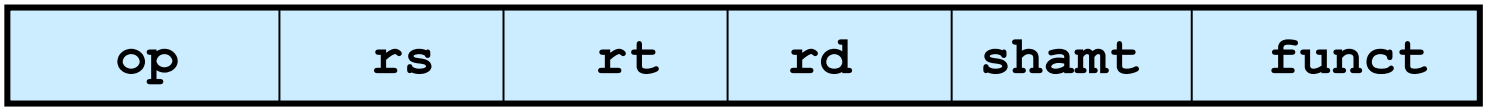
	Mnemonics	Example	Meaning
Conditional Branch	BEQ, BNE	BEQ R2, R3, -16	If R2 == R3; PC ← PC + 4 -16
Jump	J, JR	J $target_{26}$	PC ← $PC_{31\text{-}28}$ || $target_{26}$ || 00
Jump and Link	**JAL, JALR**	JAL $target_{26}$ JALR R2	**R31 ← PC + 4** **PC ← R2**
System Call	SYSCALL	SYSCALL	

MIPS-I Control Transfer Instructions

	Mnemonics	Example	Meaning
Conditional Branch	BEQ, BNE	BEQ R2, R3, -16	If R2 == R3; PC ← PC + 4 -16
Jump	J, JR	J $target_{26}$	PC ← $PC_{31\text{-}28}$ || $target_{26}$ || 00
Jump and Link	JAL, JALR	JAL $target_{26}$ JALR R2	R31 ← PC + 4 PC ← R2
System Call	**SYSCALL**	**SYSCALL**	

MIPS Instruction Formats

- R-type.



OP rd, rs, rt

op: Opcode (class of instruction). Eg. ALU
funct: Which subunit of the ALU to activate?

- I-type.

OP rt, rs, IMM

- J-type

OP LABEL

Addressing Modes

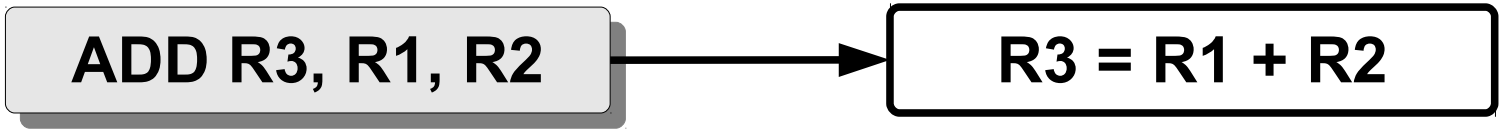
- How are the locations of operands and results specified in instructions?

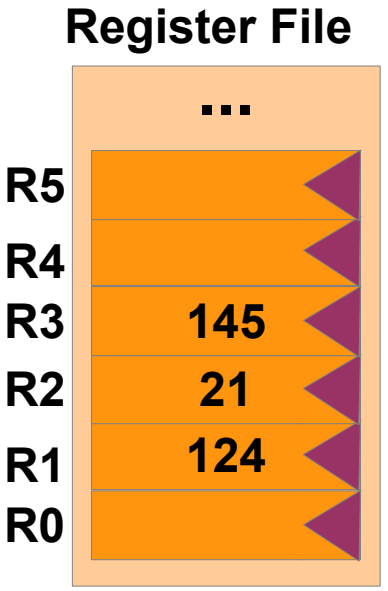
MIPS-I Instruction Set

- Addressing Modes
  - Immediate, Register (for Arithmetic and Logic instructions)
  - Absolute (for Jumps)
  - Base-displacement (for Loads, Stores)
  - PC relative (for conditional branches)

Addressing Mode Examples

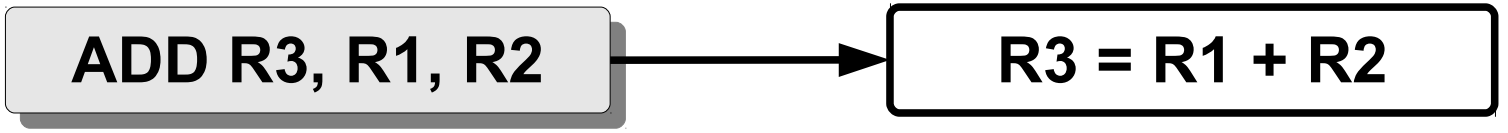
Addressing Modes

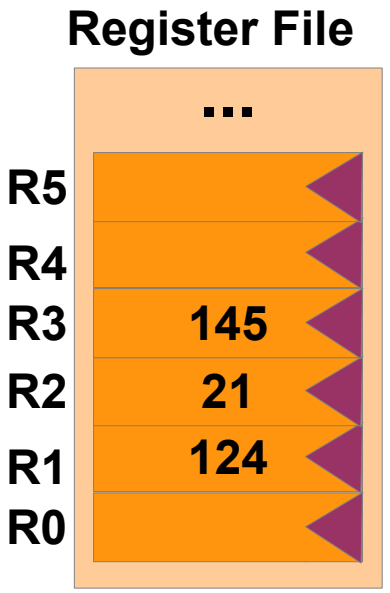




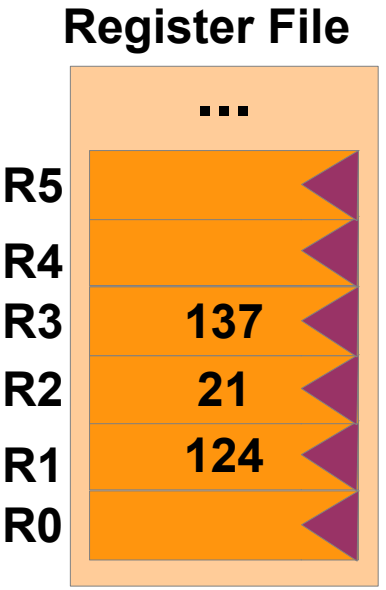
Addressing Modes

- **Register** Addressing Mode
  - Operand is in the specified general purpose register



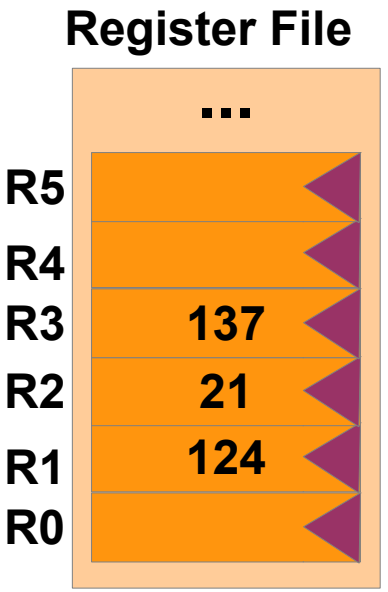


Addressing Modes

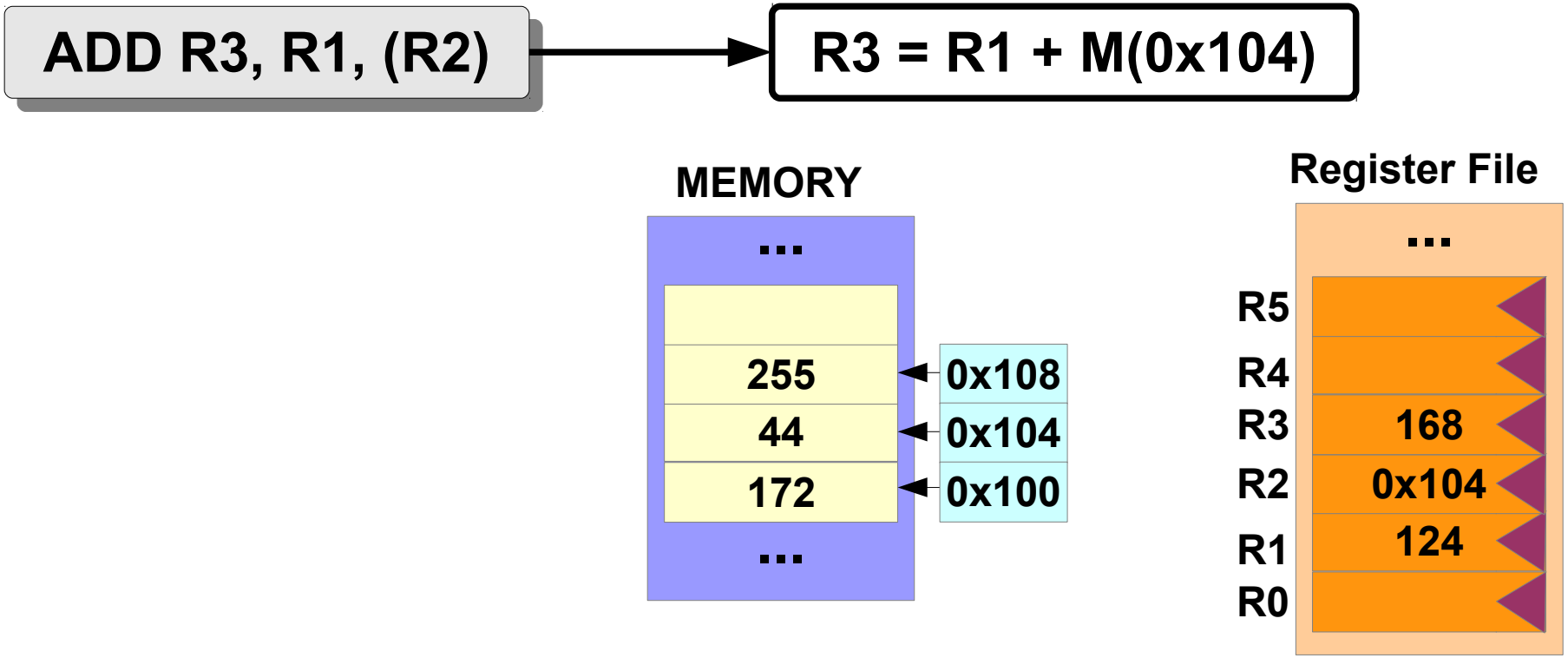


Addressing Modes

- **Immediate** Addressing Mode
  - Operand is a constant value specified inside the instruction.

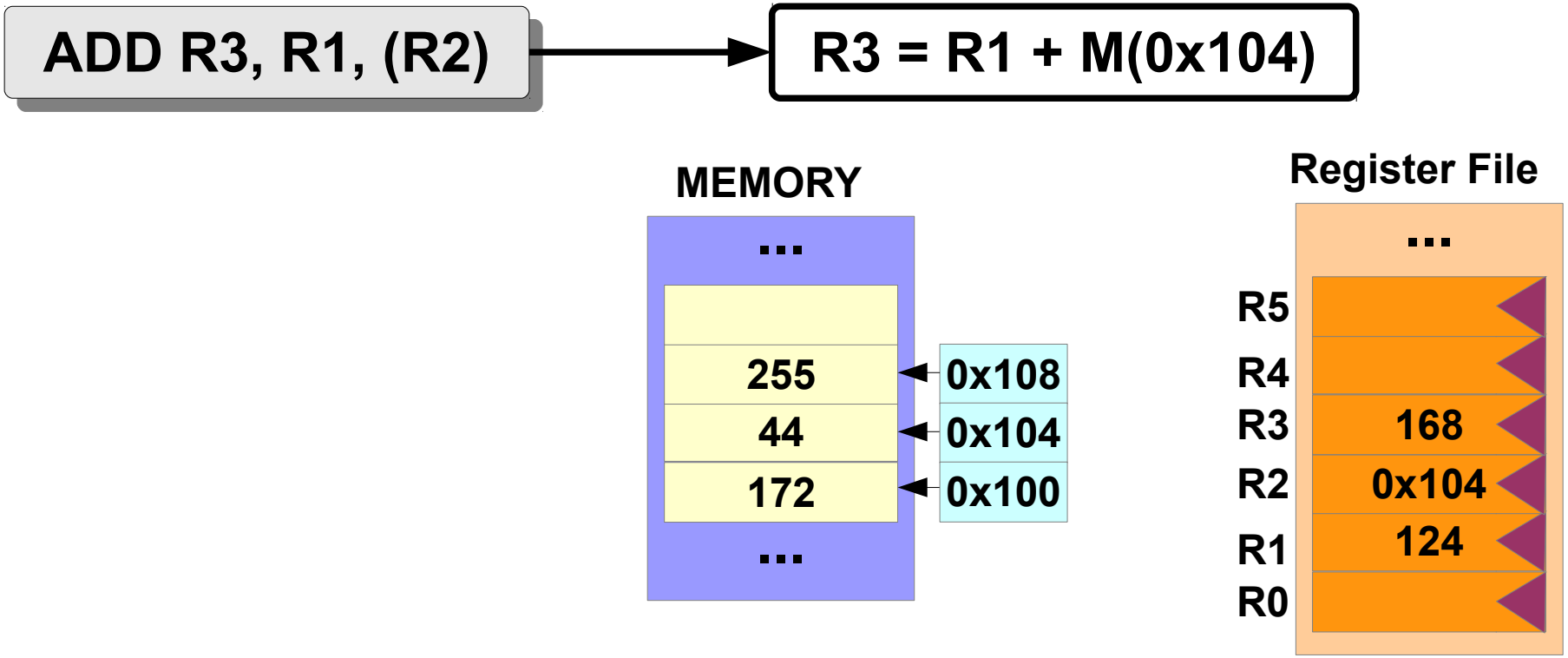


Addressing Modes

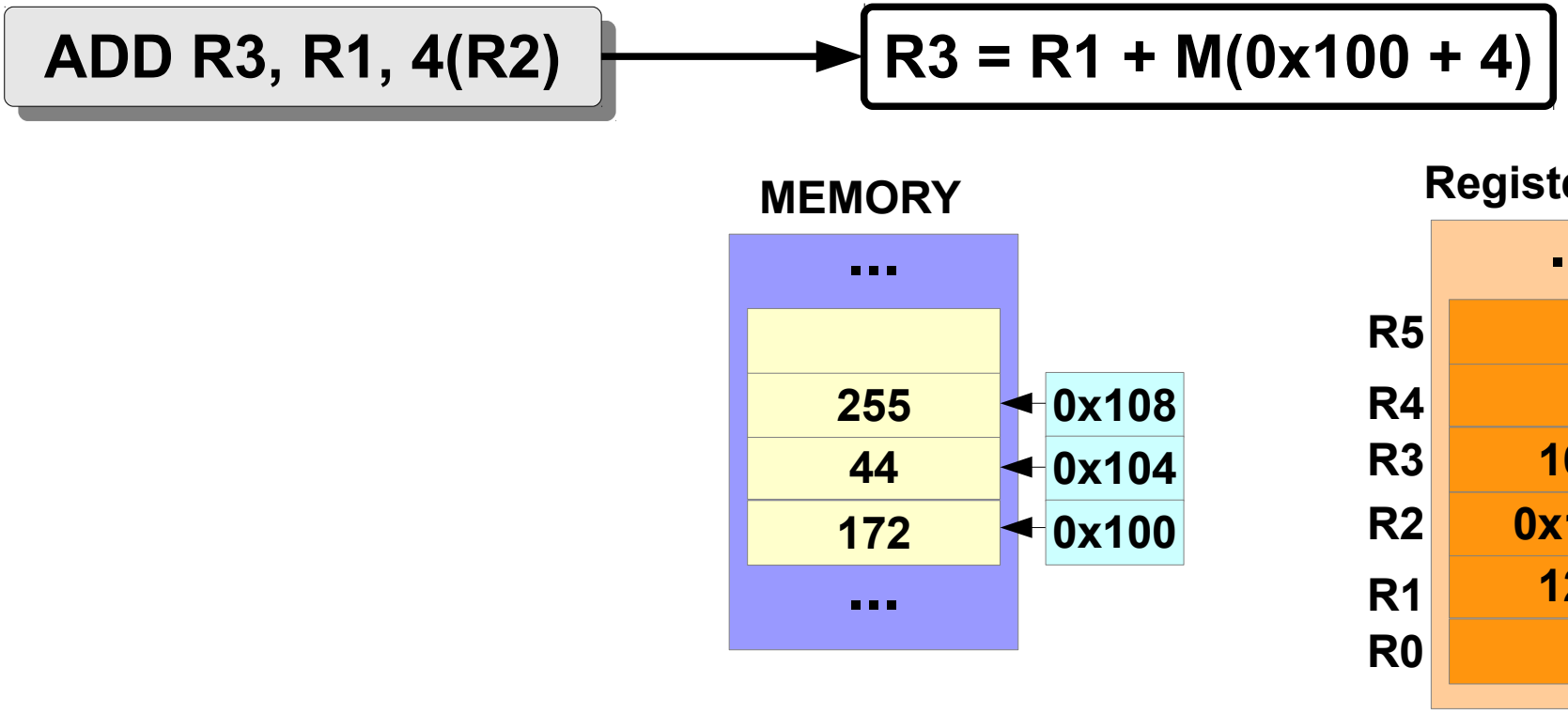


Addressing Modes

- **Register Indirect** Addressing Mode
  - Memory address of operand is in the specified general purpose register.

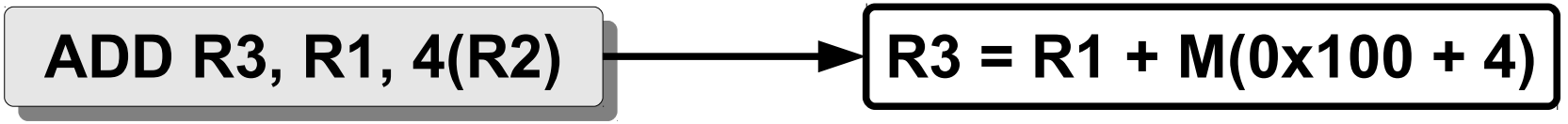


Addressing Modes

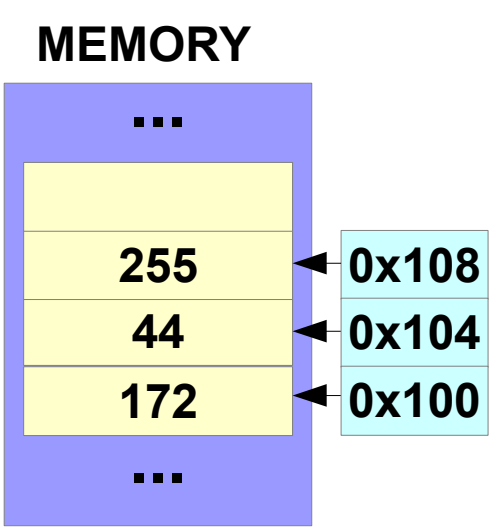


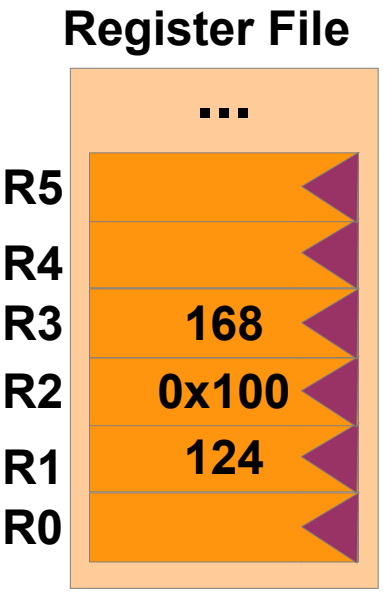
Addressing Modes

- **Base Displacement** Addressing Mode
  - Memory address of operand is calculated as the sum of value in specified register and specified displacement

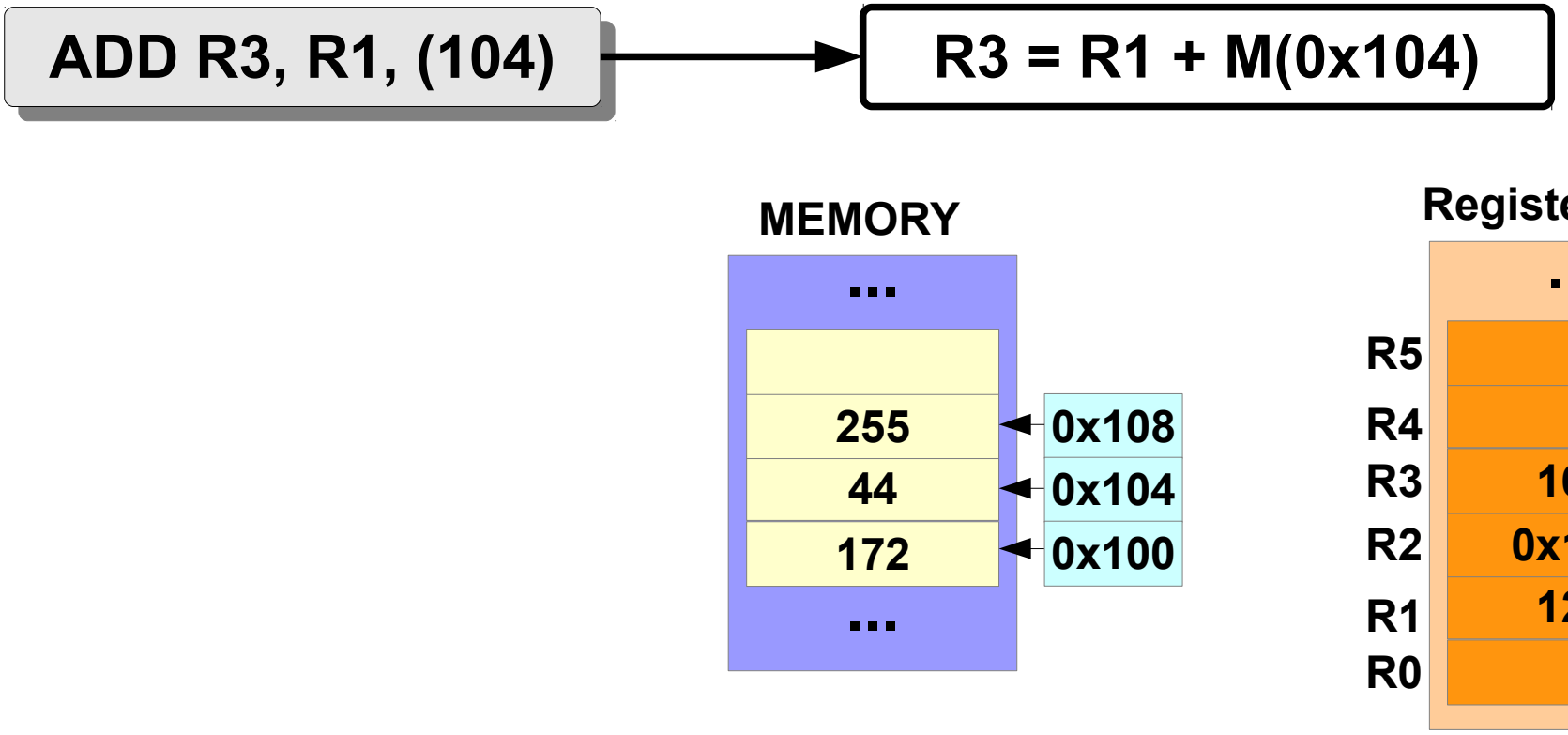


Can be used instead of Register indirect addressing mode.



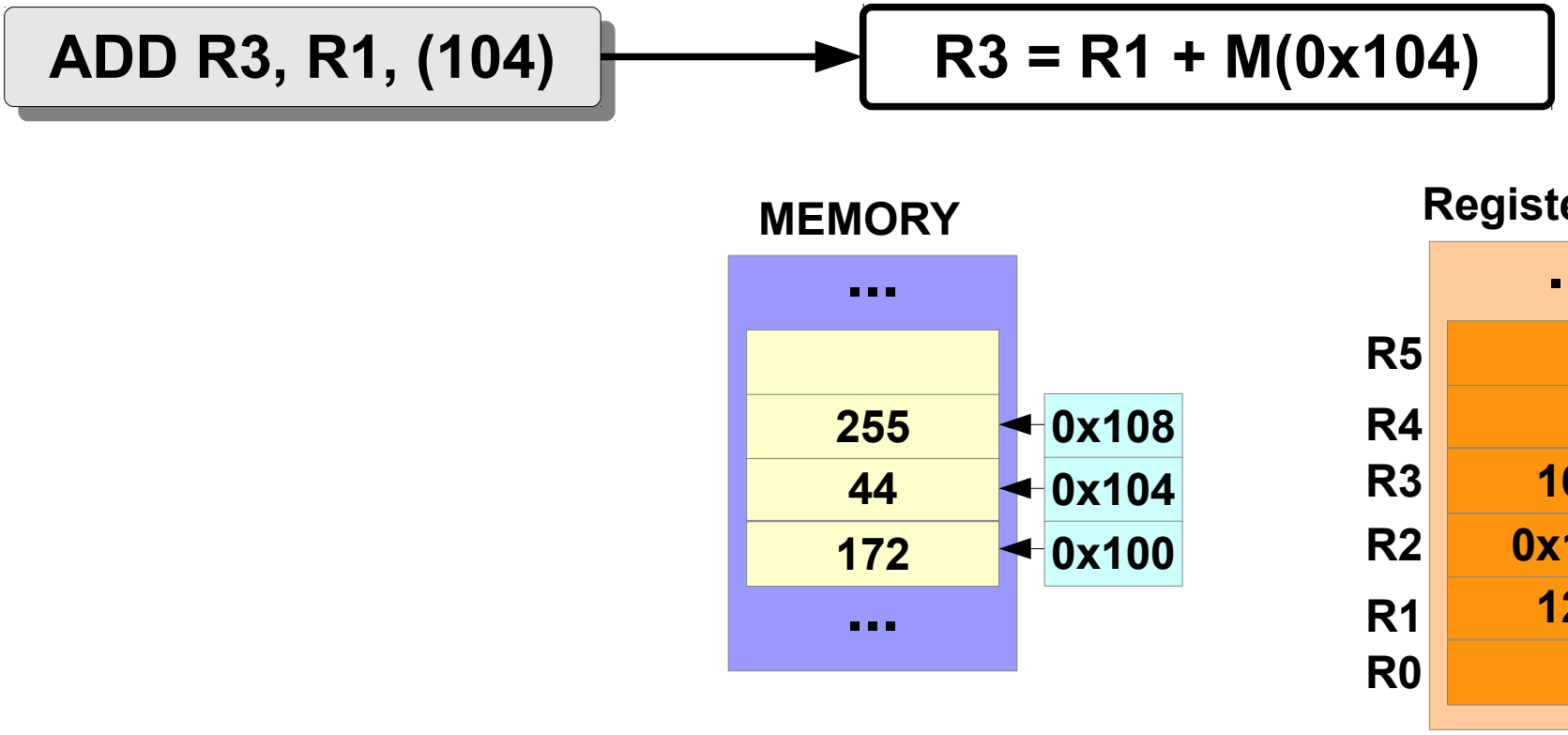


Addressing Modes



Addressing Modes

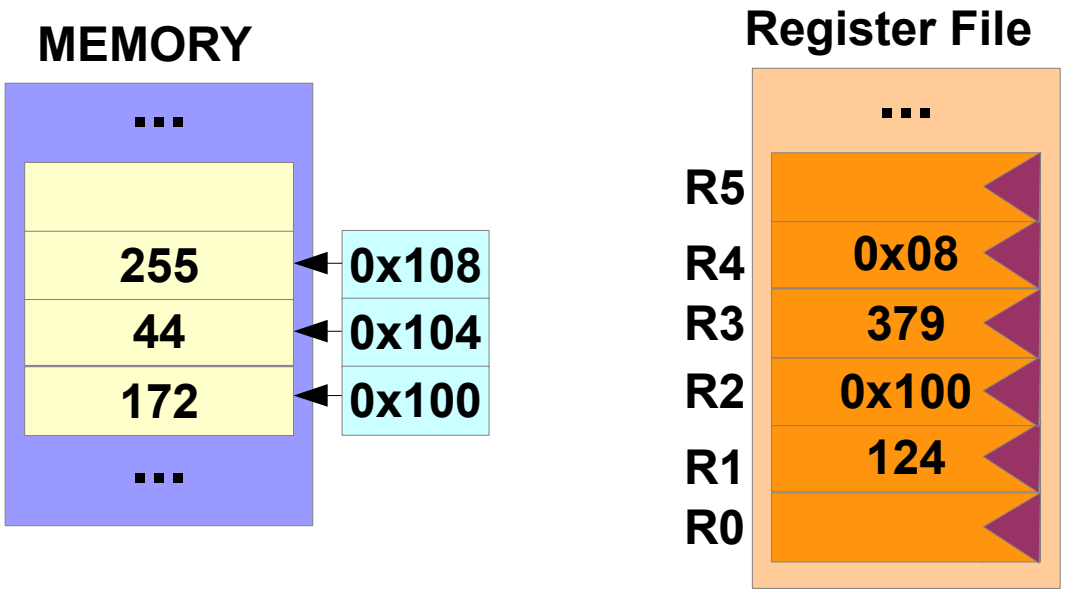
- **Absolute** Addressing Mode
  - Memory address of operand is specified directly in the instruction



Addressing Modes

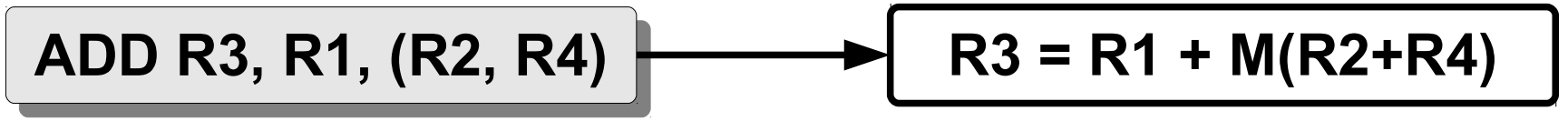
ADD R3, R1, (R2, R4) → **R3 = R1 + M(R2+R4)**

R2: Base of an array
R4: Index

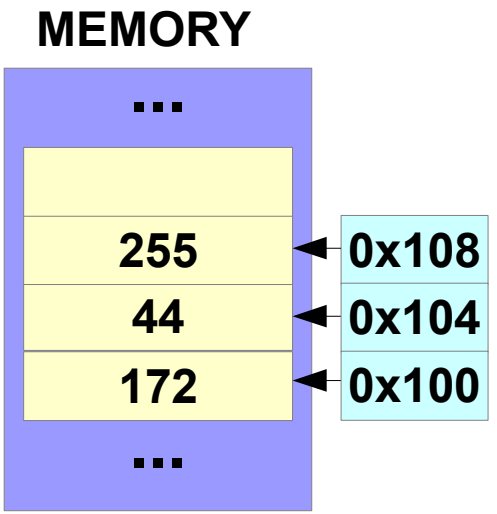


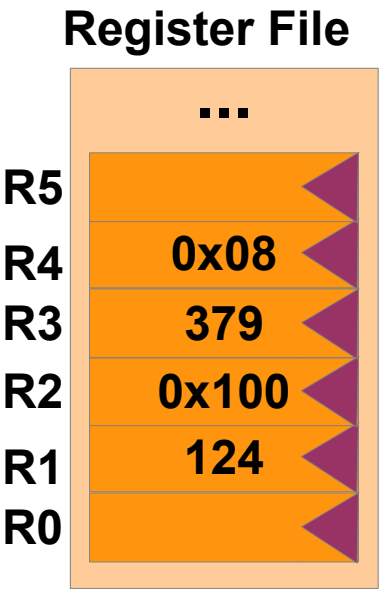
Addressing Modes

- **Indexed** Addressing Mode
  - Memory address of operand is calculated as sum of contents of 2 registers

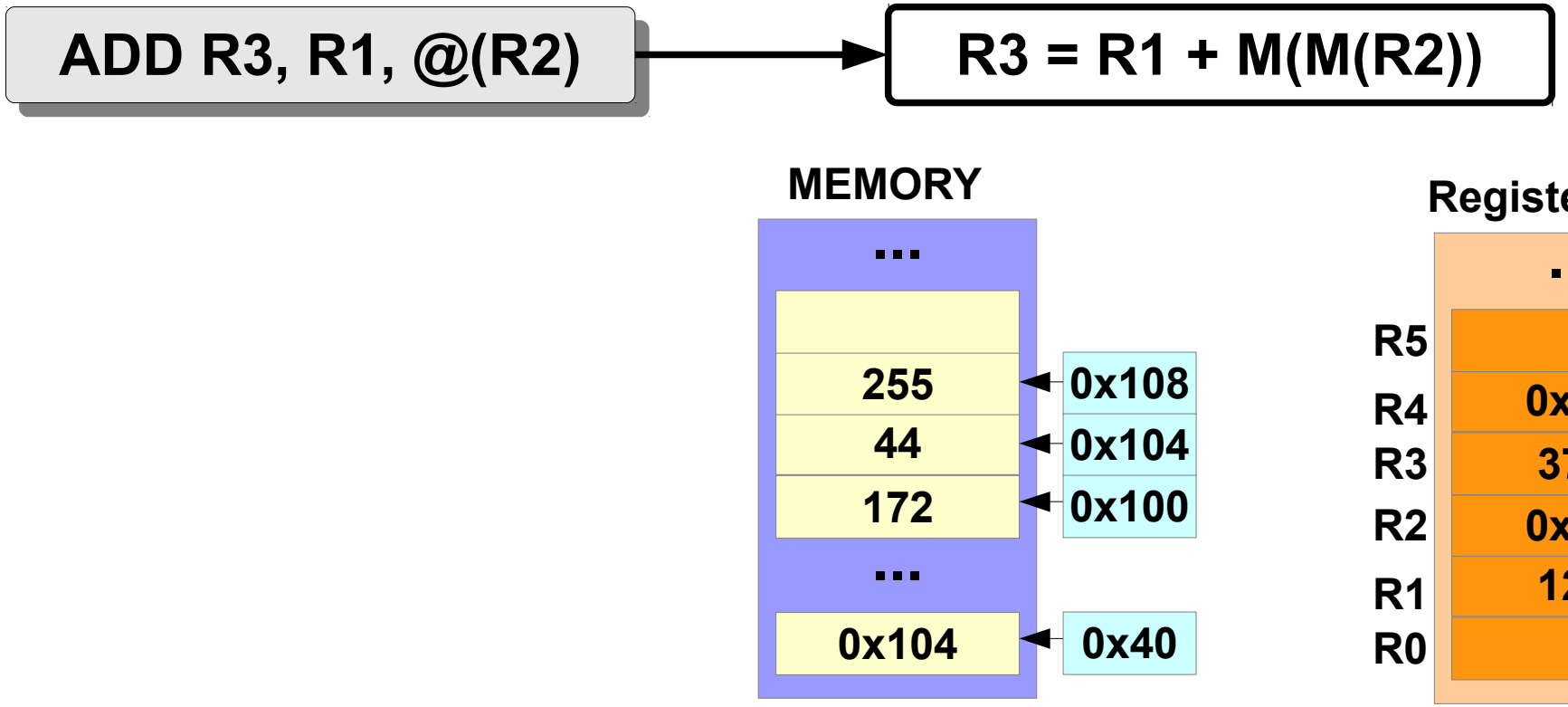


R2: Base of an array
R4: Index



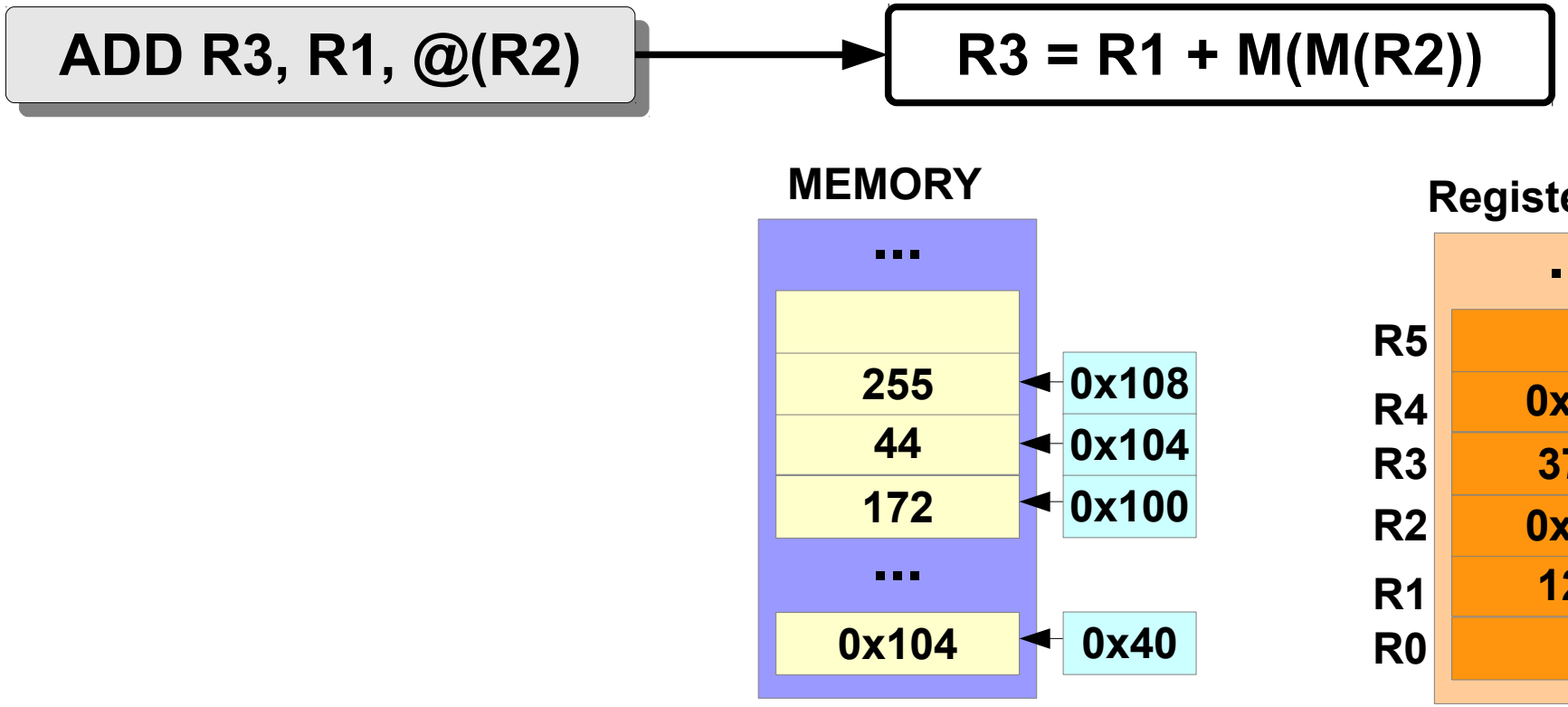


Addressing Modes

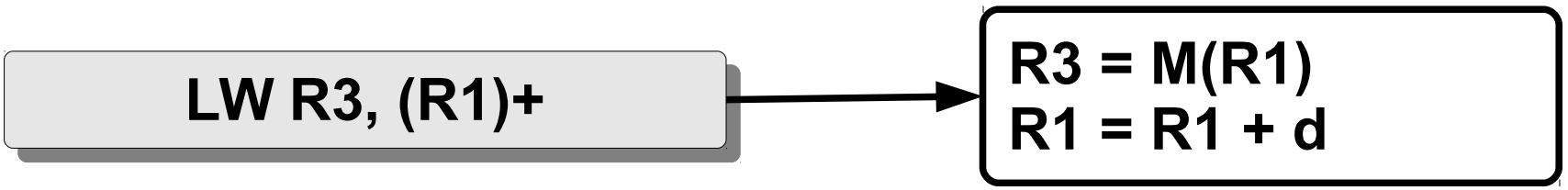


Addressing Modes

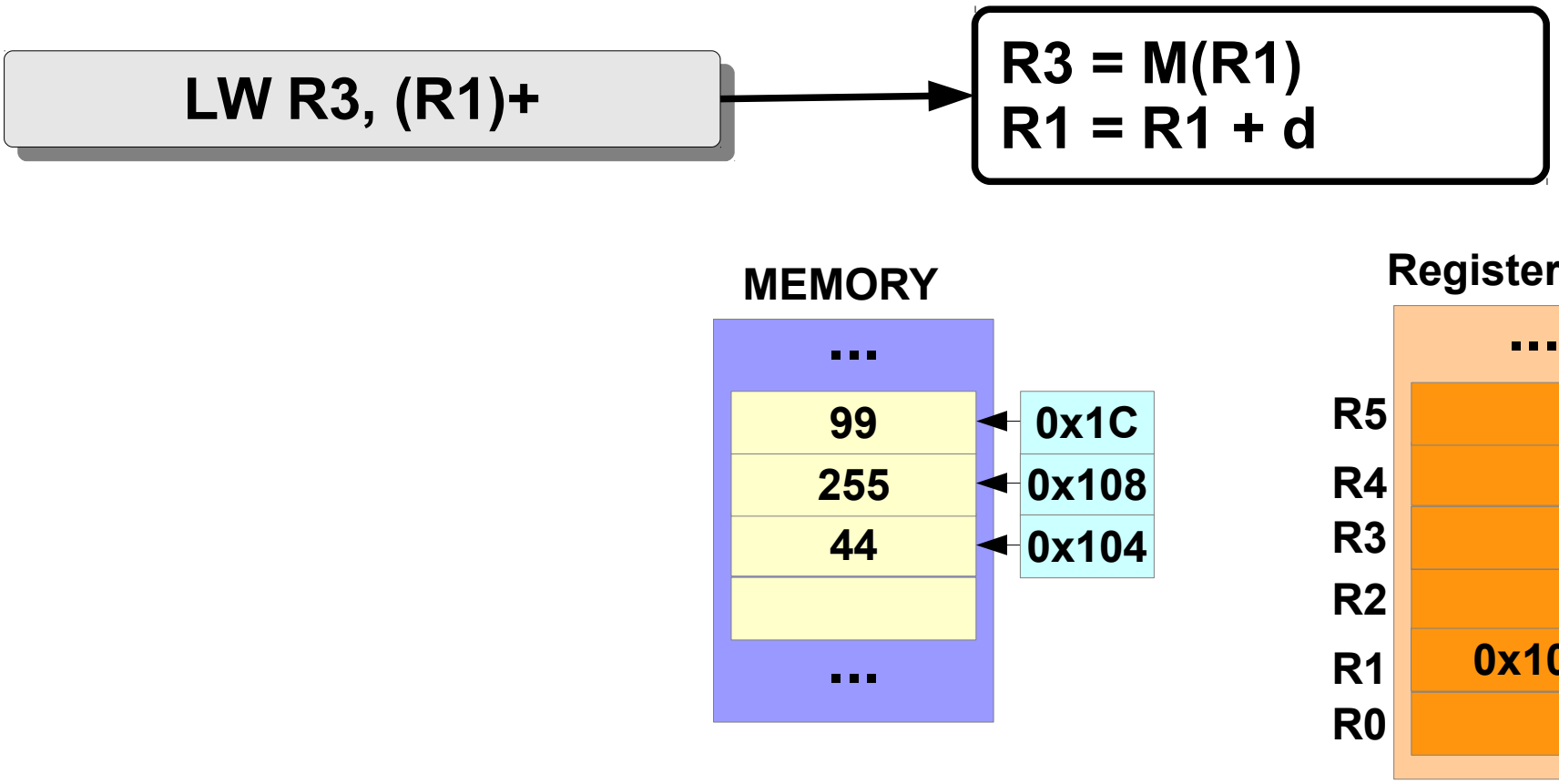
- **Memory Indirect** Addressing Mode
  - A memory location is used as a pointer to the value in another location in memory



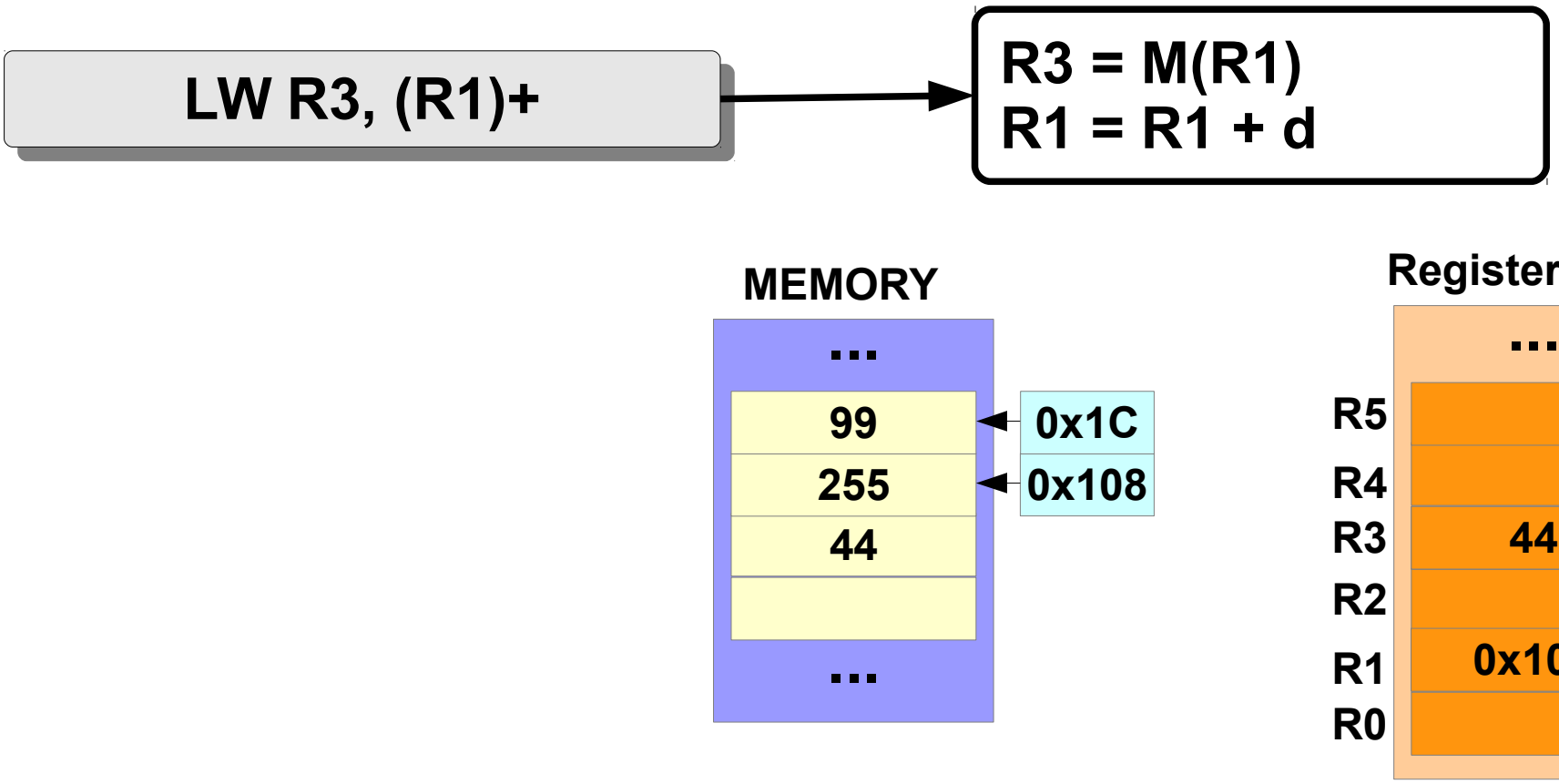
Addressing Modes



Addressing Modes

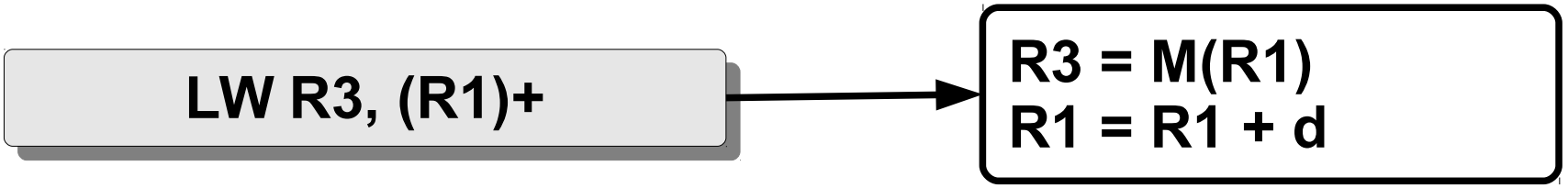


Addressing Modes

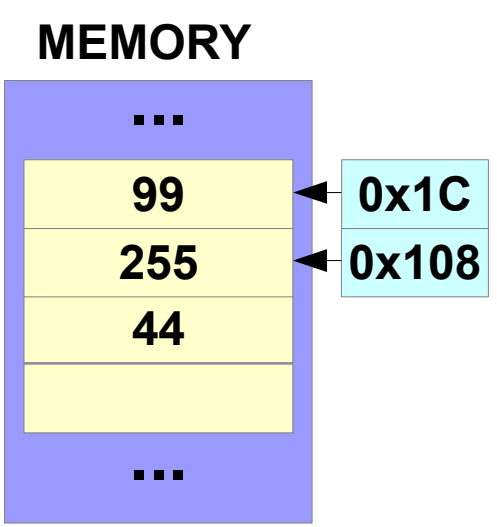


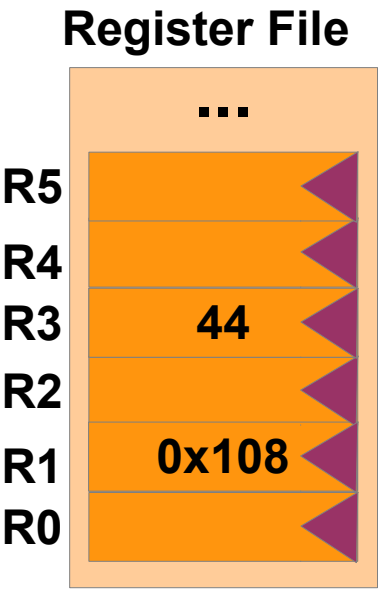
Addressing Modes

- **Auto-Increment** Addressing Mode
  - Increment the value inside a register ***after*** the operation is completed.

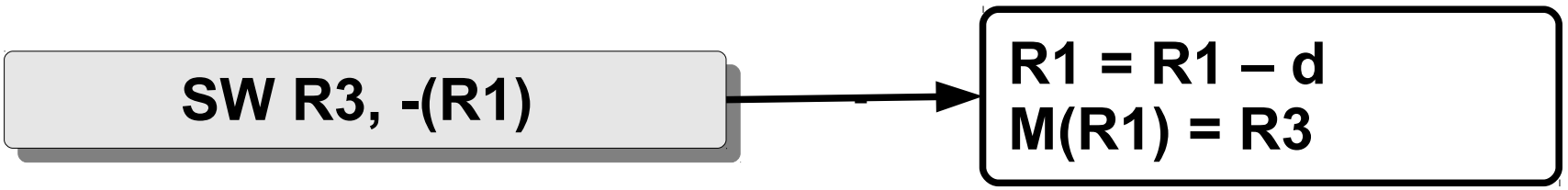


Useful for stepping through arrays within a loop.

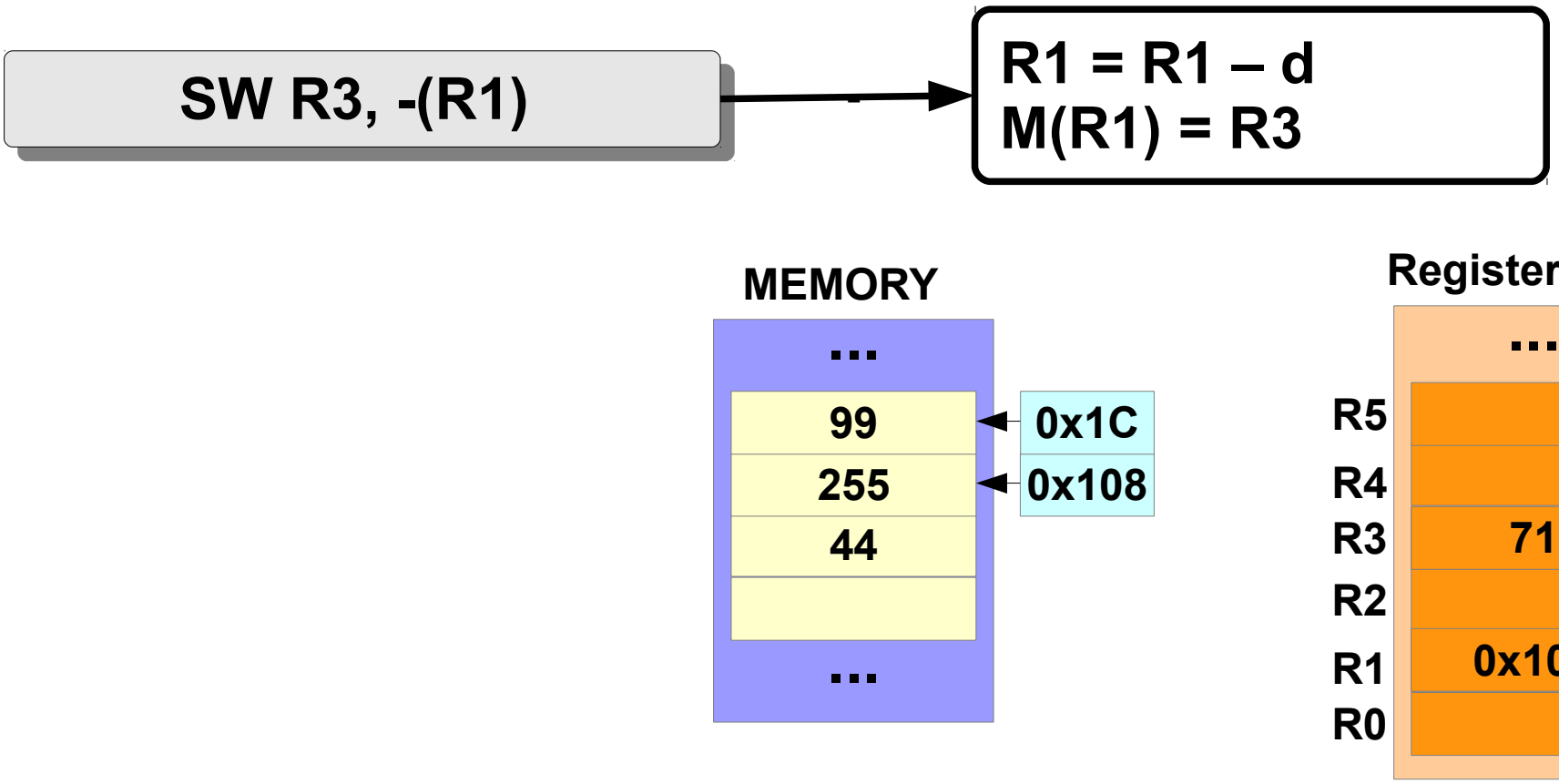




Addressing Modes

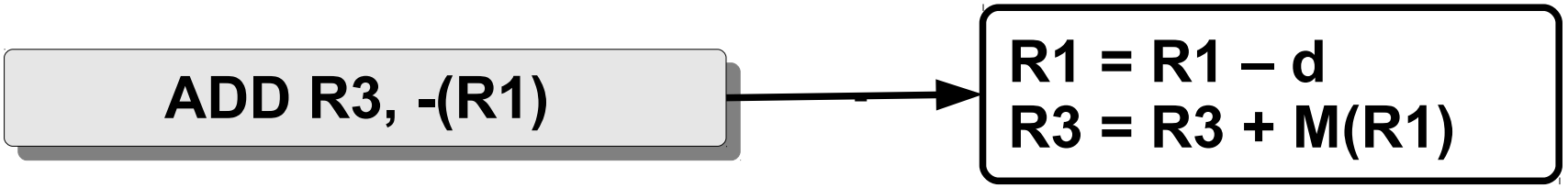


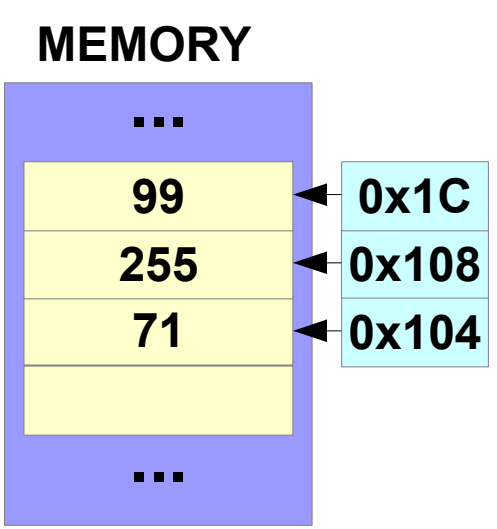
Addressing Modes

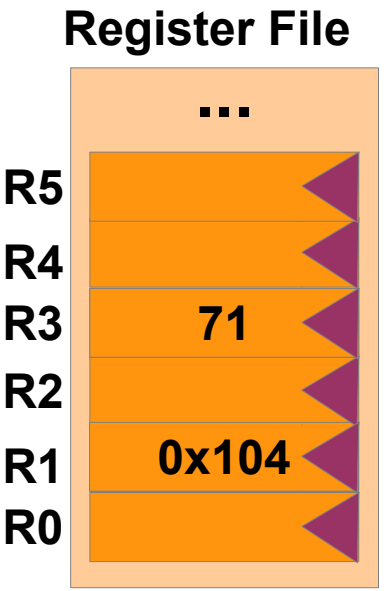


Addressing Modes

- **Auto-Decrement** Addressing Mode
  - Decrement the value inside a register ***before*** the operation is completed.

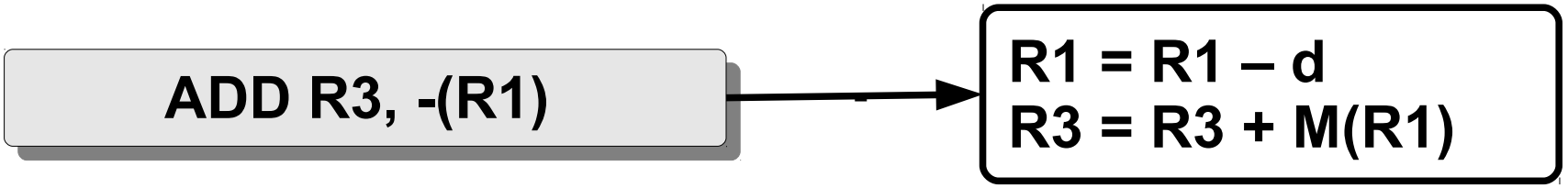




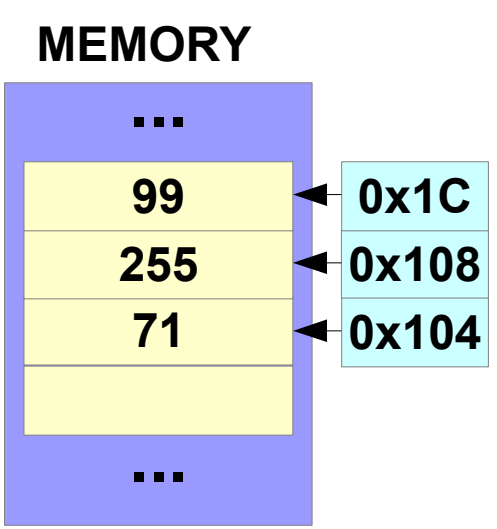


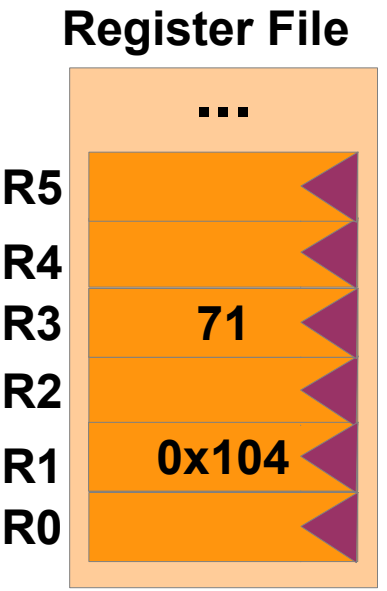
Addressing Modes

- **Auto-Decrement** Addressing Mode
  - Decrement the value inside a register ***before*** the operation is completed.

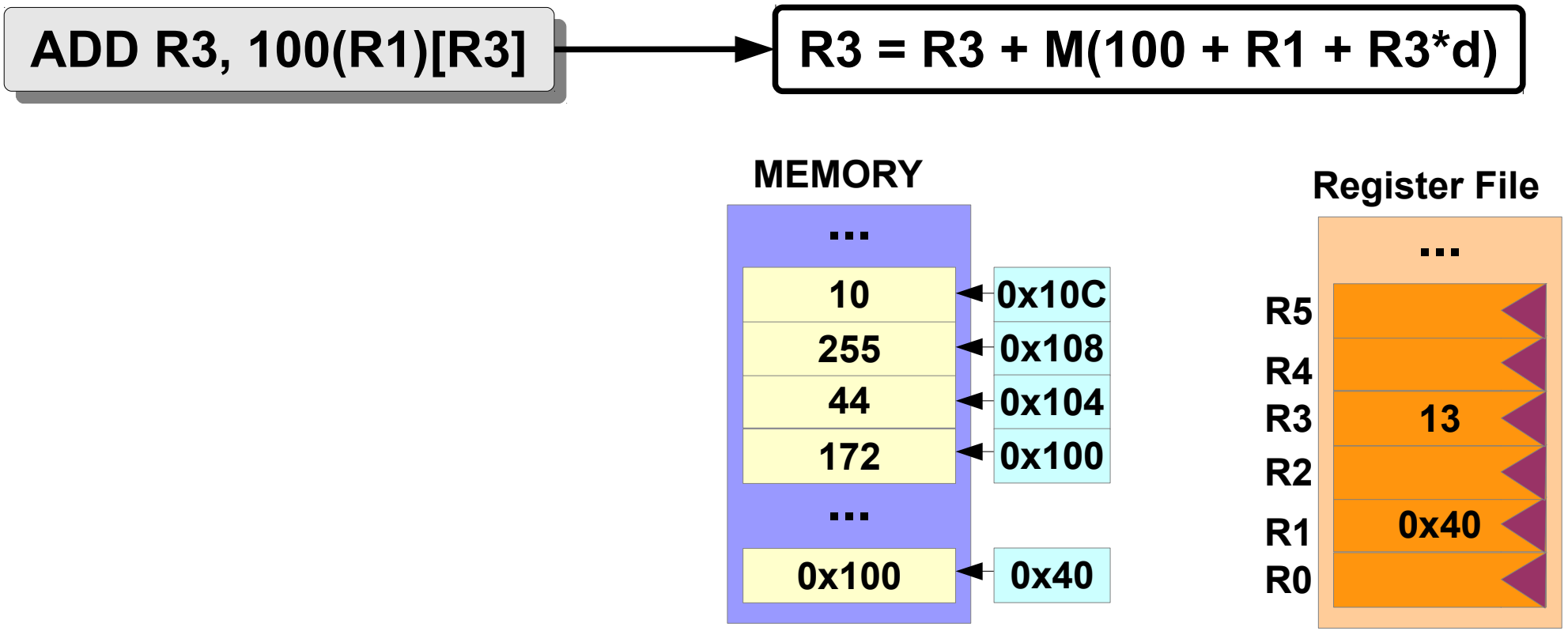


PUSH/POP operations



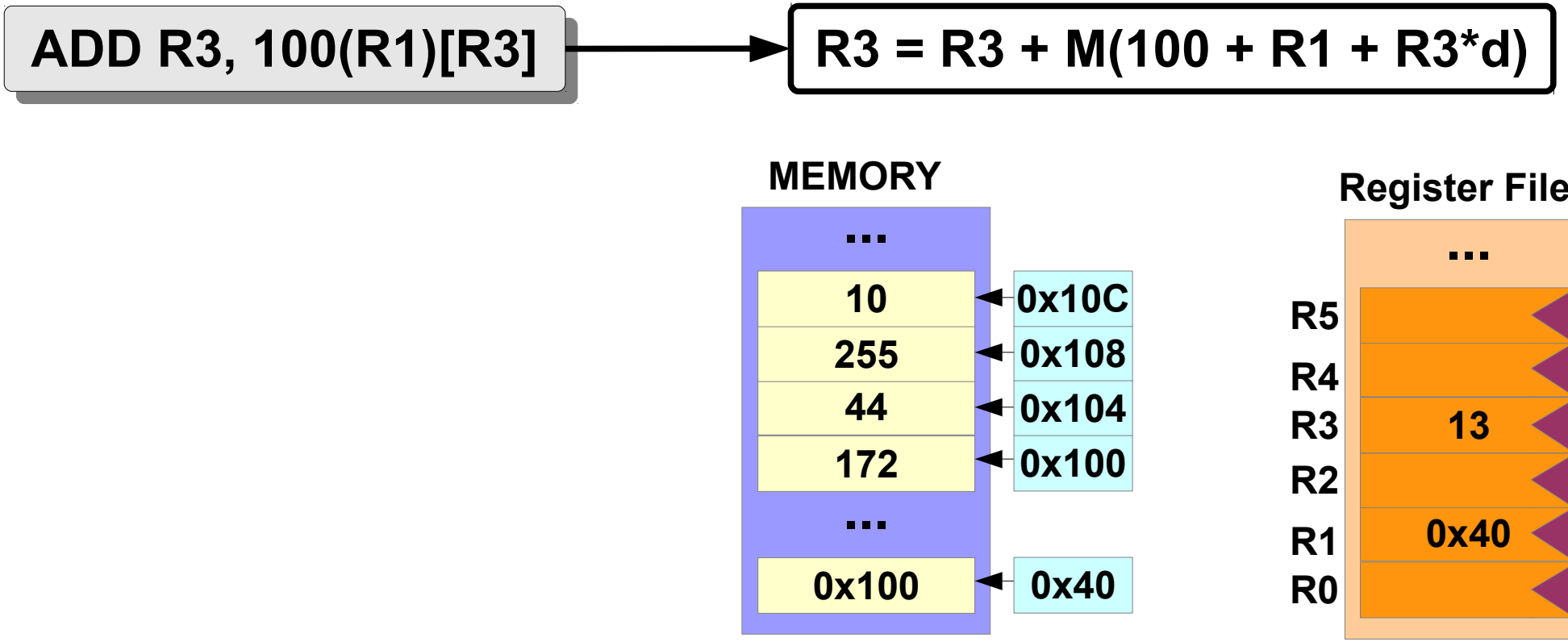


Addressing Modes

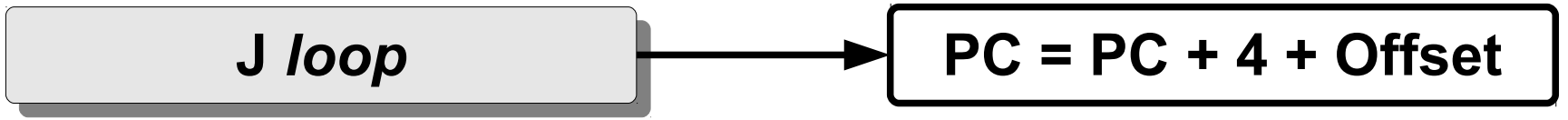


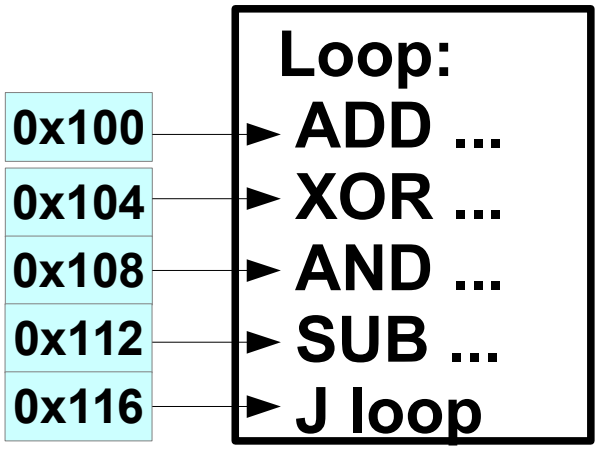
Addressing Modes

- **Scaled** Addressing Mode
  - Memory address of operand is calculated as sum of contents of 2 registers



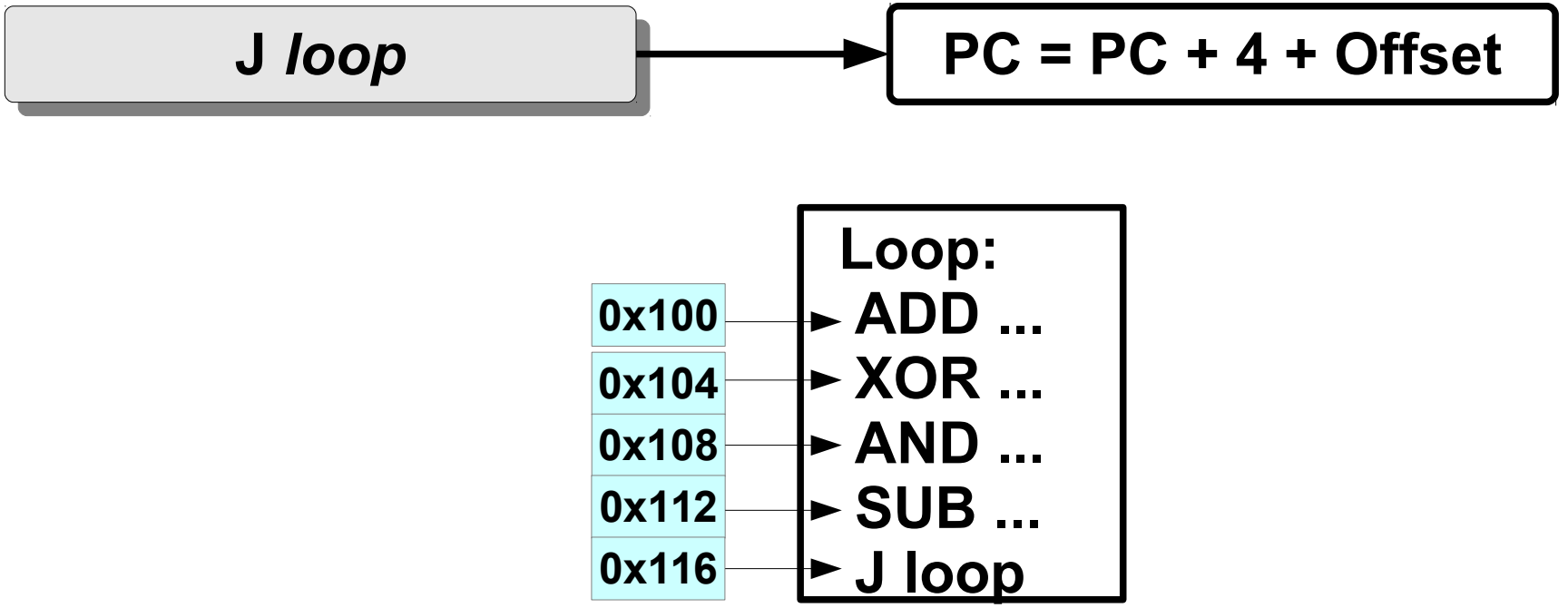
Addressing Modes





Addressing Modes

- **PC Relative** Addressing Mode
  - The operand address is specified as a displacement from the PC value (i.e., from the address of the instruction itself)



Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**
Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5	

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**
Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5	**Immediate**

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**
Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5	**Immediate**
Add R4, R3, 100(R1)	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1]]	

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**
Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5	**Immediate**
Add R4, R3, 100(R1)	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1]]	**Displacement**

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**
Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5	**Immediate**
Add R4, R3, 100(R1)	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1]]	**Displacement**
Add R4, R3, (R1)	Regs[R4] <- Regs[R3] + Mem[Regs[R1]]	

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**
Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5	**Immediate**
Add R4, R3, 100(R1)	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1]]	**Displacement**
Add R4, R3, (R1)	Regs[R4] <- Regs[R3] + Mem[Regs[R1]]	**Register Indirect**

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**
Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5	**Immediate**
Add R4, R3, 100(R1)	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1]]	**Displacement**
Add R4, R3, (R1)	Regs[R4] <- Regs[R3] + Mem[Regs[R1]]	**Register Indirect**
Add R4, R3, (0x475)	Regs[R4] <- Regs[R3] + Mem[0x475]	

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**
Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5	**Immediate**
Add R4, R3, 100(R1)	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1]]	**Displacement**
Add R4, R3, (R1)	Regs[R4] <- Regs[R3] + Mem[Regs[R1]]	**Register Indirect**
Add R4, R3, (0x475)	Regs[R4] <- Regs[R3] + Mem[0x475]	**Absolute**

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**
Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5	**Immediate**
Add R4, R3, 100(R1)	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1]]	**Displacement**
Add R4, R3, (R1)	Regs[R4] <- Regs[R3] + Mem[Regs[R1]]	**Register Indirect**
Add R4, R3, (0x475)	Regs[R4] <- Regs[R3] + Mem[0x475]	**Absolute**
Add R4, R3, @(R1)	Regs[R4] <- Regs[R3] + Mem[Mem[R1]]	

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**
Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5	**Immediate**
Add R4, R3, 100(R1)	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1]]	**Displacement**
Add R4, R3, (R1)	Regs[R4] <- Regs[R3] + Mem[Regs[R1]]	**Register Indirect**
Add R4, R3, (0x475)	Regs[R4] <- Regs[R3] + Mem[0x475]	**Absolute**
Add R4, R3, @(R1)	Regs[R4] <- Regs[R3] + Mem[Mem[R1]]	**Memory Indirect**

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**
Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5	**Immediate**
Add R4, R3, 100(R1)	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1]]	**Displacement**
Add R4, R3, (R1)	Regs[R4] <- Regs[R3] + Mem[Regs[R1]]	**Register Indirect**
Add R4, R3, (0x475)	Regs[R4] <- Regs[R3] + Mem[0x475]	**Absolute**
Add R4, R3, @(R1)	Regs[R4] <- Regs[R3] + Mem[Mem[R1]]	**Memory Indirect**
Add R4, R3, 100(PC)	Regs[R4] <- Regs[R3] + Mem[100 + PC]	

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**
Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5	**Immediate**
Add R4, R3, 100(R1)	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1]]	**Displacement**
Add R4, R3, (R1)	Regs[R4] <- Regs[R3] + Mem[Regs[R1]]	**Register Indirect**
Add R4, R3, (0x475)	Regs[R4] <- Regs[R3] + Mem[0x475]	**Absolute**
Add R4, R3, @(R1)	Regs[R4] <- Regs[R3] + Mem[Mem[R1]]	**Memory Indirect**
Add R4, R3, 100(PC)	Regs[R4] <- Regs[R3] + Mem[100 + PC]	**PC relative**

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**
Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5	**Immediate**
Add R4, R3, 100(R1)	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1]]	**Displacement**
Add R4, R3, (R1)	Regs[R4] <- Regs[R3] + Mem[Regs[R1]]	**Register Indirect**
Add R4, R3, (0x475)	Regs[R4] <- Regs[R3] + Mem[0x475]	**Absolute**
Add R4, R3, @(R1)	Regs[R4] <- Regs[R3] + Mem[Mem[R1]]	**Memory Indirect**
Add R4, R3, 100(PC)	Regs[R4] <- Regs[R3] + Mem[100 + PC]	**PC relative**
Add R4, R3, 100(R1)[R5]	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1] + Regs[R5] * 4]	

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**
Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5	**Immediate**
Add R4, R3, 100(R1)	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1]]	**Displacement**
Add R4, R3, (R1)	Regs[R4] <- Regs[R3] + Mem[Regs[R1]]	**Register Indirect**
Add R4, R3, (0x475)	Regs[R4] <- Regs[R3] + Mem[0x475]	**Absolute**
Add R4, R3, @(R1)	Regs[R4] <- Regs[R3] + Mem[Mem[R1]]	**Memory Indirect**
J loop	PC = PC + 4 + Offset	**PC relative**
Add R4, R3, 100(R1)[R5]	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1] + Regs[R5] * 4]	**Scaled**

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**
Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5	**Immediate**
Add R4, R3, 100(R1)	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1]]	**Displacement**
Add R4, R3, (R1)	Regs[R4] <- Regs[R3] + Mem[Regs[R1]]	**Register Indirect**
Add R4, R3, (0x475)	Regs[R4] <- Regs[R3] + Mem[0x475]	**Absolute**
Add R4, R3, @(R1)	Regs[R4] <- Regs[R3] + Mem[Mem[R1]]	**Memory Indirect**
J loop	PC = PC + 4 + Offset	**PC relative**
Add R4, R3, 100(R1)[R5]	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1] + Regs[R5] * 4]	**Scaled**
LW R4, (R3)+	Regs[R4] <- Regs[R4] + Mem[R3] Regs[R3] <- Regs[R3] + d	

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**
Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5	**Immediate**
Add R4, R3, 100(R1)	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1]]	**Displacement**
Add R4, R3, (R1)	Regs[R4] <- Regs[R3] + Mem[Regs[R1]]	**Register Indirect**
Add R4, R3, (0x475)	Regs[R4] <- Regs[R3] + Mem[0x475]	**Absolute**
Add R4, R3, @(R1)	Regs[R4] <- Regs[R3] + Mem[Mem[R1]]	**Memory Indirect**
J loop	PC = PC + 4 + Offset	**PC relative**
Add R4, R3, 100(R1)[R5]	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1] + Regs[R5] * 4]	**Scaled**
LW R4, (R3)+	Regs[R4] <- Regs[R4] + Mem[R3] Regs[R3] <- Regs[R3] + d	**Auto Increment**

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**
Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5	**Immediate**
Add R4, R3, 100(R1)	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1]]	**Displacement**
Add R4, R3, (R1)	Regs[R4] <- Regs[R3] + Mem[Regs[R1]]	**Register Indirect**
Add R4, R3, (0x475)	Regs[R4] <- Regs[R3] + Mem[0x475]	**Absolute**
Add R4, R3, @(R1)	Regs[R4] <- Regs[R3] + Mem[Mem[R1]]	**Memory Indirect**
J loop	PC = PC + 4 + Offset	**PC relative**
Add R4, R3, 100(R1)[R5]	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1] + Regs[R5] * 4]	**Scaled**
LW R4, (R3)+	Regs[R4] <- Regs[R4] + Mem[R3] Regs[R3] <- Regs[R3] + d	**Auto Increment**
SW R4, -(R3)	Regs[R3] <- Regs[R3] - d Regs[R4] <- Regs[R4] + Mem[R3]	

Addressing Modes

Add R1, R2, R3	Regs[R4] <- Regs[R3] + Regs[R2]	**Register**
Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5	**Immediate**
Add R4, R3, 100(R1)	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1]]	**Displacement**
Add R4, R3, (R1)	Regs[R4] <- Regs[R3] + Mem[Regs[R1]]	**Register Indirect**
Add R4, R3, (0x475)	Regs[R4] <- Regs[R3] + Mem[0x475]	**Absolute**
Add R4, R3, @(R1)	Regs[R4] <- Regs[R3] + Mem[Mem[R1]]	**Memory Indirect**
J loop	PC = PC + 4 + Offset	**PC relative**
Add R4, R3, 100(R1)[R5]	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1] + Regs[R5] * 4]	**Scaled**
LW R4, (R3)+	Regs[R4] <- Regs[R4] + Mem[R3] Regs[R3] <- Regs[R3] + d	**Auto Increment**
SW R4, -(R3)	Regs[R3] <- Regs[R3] - d Regs[R4] <- Regs[R4] + Mem[R3]	**Auto Decrement**

Addressing Modes

- Which ones should a new architecture support?
- What should be size of the displacement?
- What should be size of the immediate field?

Addressing Modes

- Which ones should a new architecture support?
- What should be size of the displacement?
- What should be size of the immediate field?
- Benchmarks: GCC, Particle simulations (Physics, Chemistry), Large databases, Video encoding/decoding
- Popular Choices:
  - Displacement, Immediate, Register Indirect
  - 12 – 16b

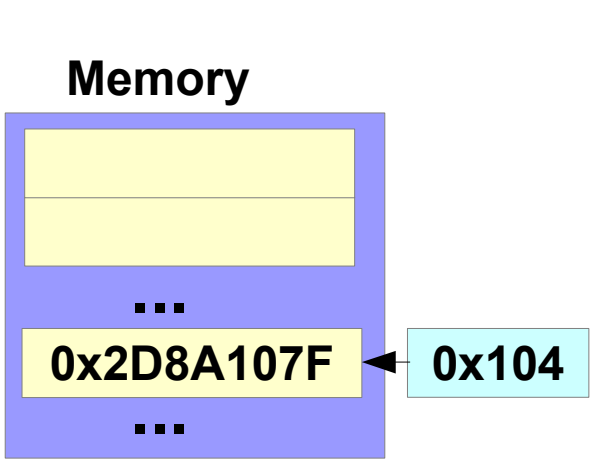
Module Outline

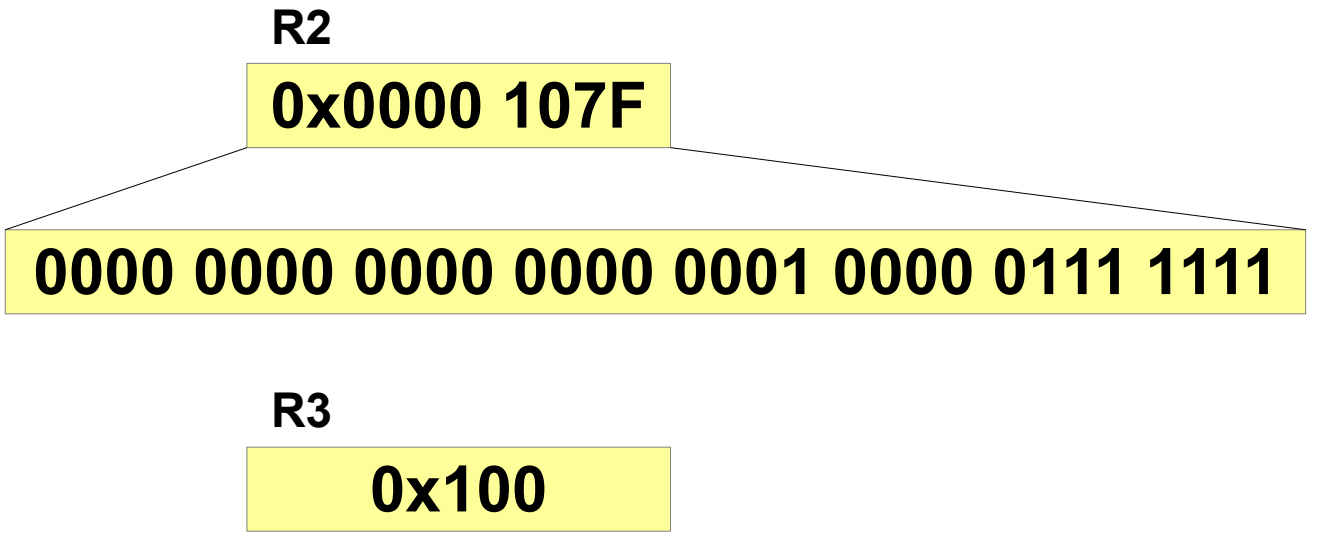
- Instruction classes.
- MIPS-I ISA.
- Addressing modes
- High level languages, Assembly languages and object code.
- Subroutine and subroutine call.
- Translating and starting a program.

Extra Slides

MIPS-I Data Transfer Instructions

	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	LB R2, 4(R3) **LH R2, 4(R3)** LW R2, 4(R3) LBU R2, 4(R3) LUI R2, 4(R3)	$R2_{15\text{-}0} \leftarrow Mem(R3 + 4)_{15\text{-}0}$ $R2_{31\text{-}16} \leftarrow$ Sign Extension





MIPS-I Data Transfer Instructions

	Mnemonics	Example	Meaning
Load	LB, LBU, LH, LHU, LW, LUI	LW R2, 4(R3)	R2 ← Mem(R3 + 4)
Store	SB, SH, SW	SB R2, -8(R4)	Mem(R4 - 8) ← R2
Move	**MFHI, MFLO, MTHI, MTLO**	**MFHI R1**	**R2 ← HI**

- L: Load
- S: Store
- M: Move from/to HI/LO
- B: Byte (8b), H: Half Word (16b), W: Word (32b)
- U: Upper
- I: Immediate

MIPS-I Arithmetic Instructions

	Mnemonics	Example	Meaning
Add, Subtract	ADD, ADDU, ADDI, ADDIU, SUB, SUBU	ADD R1, R2, R3 ADDI R1, R2, 6	R1 ← R2 + R3 R1 ← R2 + 6
Multiply, Divide	MULT, DIV, MULTU, DIVU	MULT R1, R2	LO ← lsw (R1 * R2) HI ← msw (R1 * R2)

- Divide Operation
  - Quotient in LO, Remainder in HI

x86 (IA-32) Instruction Encoding

Instruction Prefix	Opcode	ModR/M	Scale,Index Base	Displace ment	Immediate
Up to four prefixes (1 byte each)	1, 2 or 3B	1B (if needed)	1B (if needed)	0,1,2, or 4B (if needed)	0,1,2, or 4B (if needed)

REP MOVSB

x86 and x86-64 instruction format
Possible instructions 1 to 18 bytes long

ISA – Examples

Arithmetic Example

- C code:

```
f = (g + h) - (i + j);
```

  - {f,g,h,i,j} in {$s0,$s1,$s2,$s3,$s4}

- Compiled MIPS code:

```
add $t0, $s1, $s2
add $t1, $s3, $s4
sub $s0, $t0, $t1
```

R-format Example

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

```
add $t0, $s1, $s2
```

special	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

$00000010001100100100000000100000_2 = 02324020_{16}$

I-format Example


```
addi $t0, $s1, 10
```

addi	$s1	$t0	10
0	17	8	10
001000	10001	01000	0000 0000 0000 1010

$$00100001000101000000000000001010_2 = 2228000A_{16}$$