Computer Architecture (ENE1004)

▼ Lec 3

Lec 3: Instructions: Language of the Computer 2

Review: MIPS Instructions

- MIPS Instructions: assembly instructions (or its corresponding binary values)
- Arithmetic instructions
  - Three operands, each of which can be register or constant

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1,$s2,$s3`	$s1 = $s2 + $s3	Three register operands
	subtract	`sub $s1,$s2,$s3`	$s1 = $s2 – $s3	Three register operands
	add immediate	`addi $s1,$s2,20`	$s1 = $s2 + 20	Used to add constants

- Data transfer instructions
  - Moving data between a register within CPU and a location of the memory
  - Two operands; one is a register and the other is the memory location

Data transfer	load word	`lw $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	`sw $s1,20($s2)`	Memory[$s2 + 20] = $s1	Word from register to memory

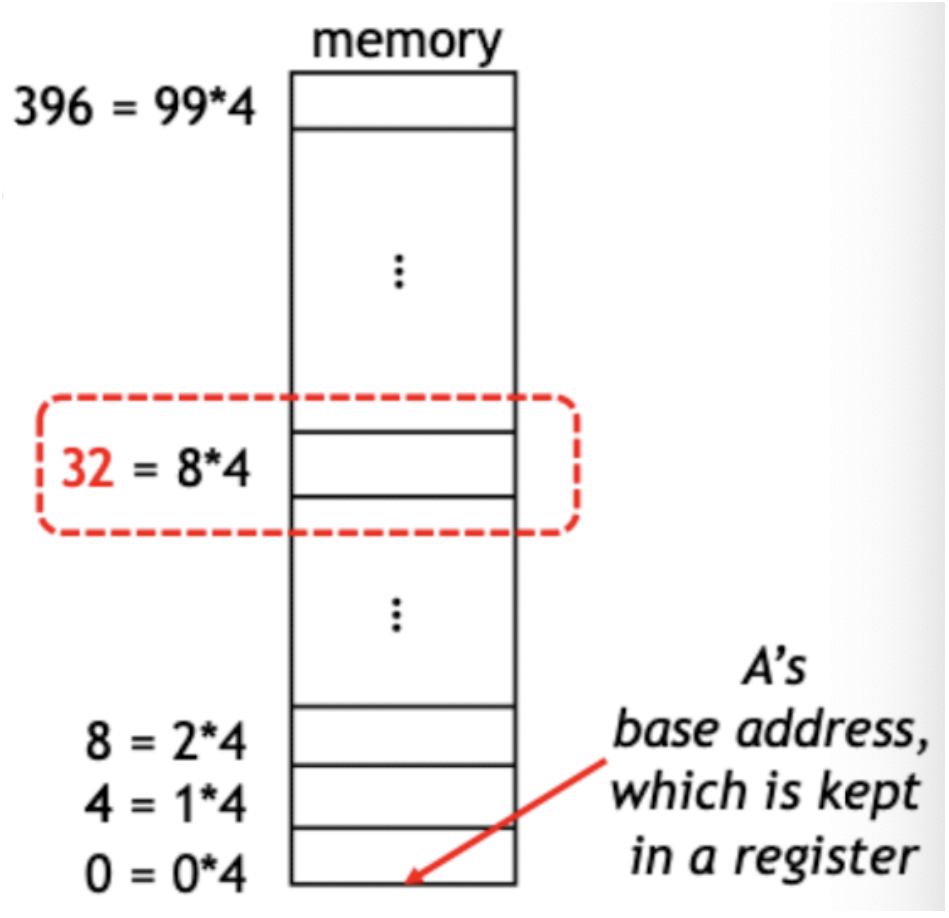
Review: Load Word Instruction

lw: 메모리 위치에서 레지스터로 데이터를 전송하는 명령어

- lw (load word)
  - lw register_to_be_loaded memory_address_to_be_accessed (lw “로드 될 레지스터” “접근 할 메모리 주소”)
  - memory_address_to_be_accessed: index (register that holds the base address of memory: 메모리의 기본 주소를 보유한 레지스터)

    - Here, index must be specified in bytes
- Example
    - Assumption 1: A is an (4-byte) integer array of 100 elements
    - Assumption 2: Compiler associates variables g and h with $s1 and $s2
    - Assumption 3: starting(base) address of array A is in $s3
    - C code: g = h + A[8]; is translated into (1개의 operation으로 계산할 수 있지만 A[8]이 array라서 load로 데이터를 받아와야 함.)

```
lw $t0, 32($s3) # temporary reg $t0 gets A[8]
add $s1, $s2, $t0 # g = h + A[8]
```



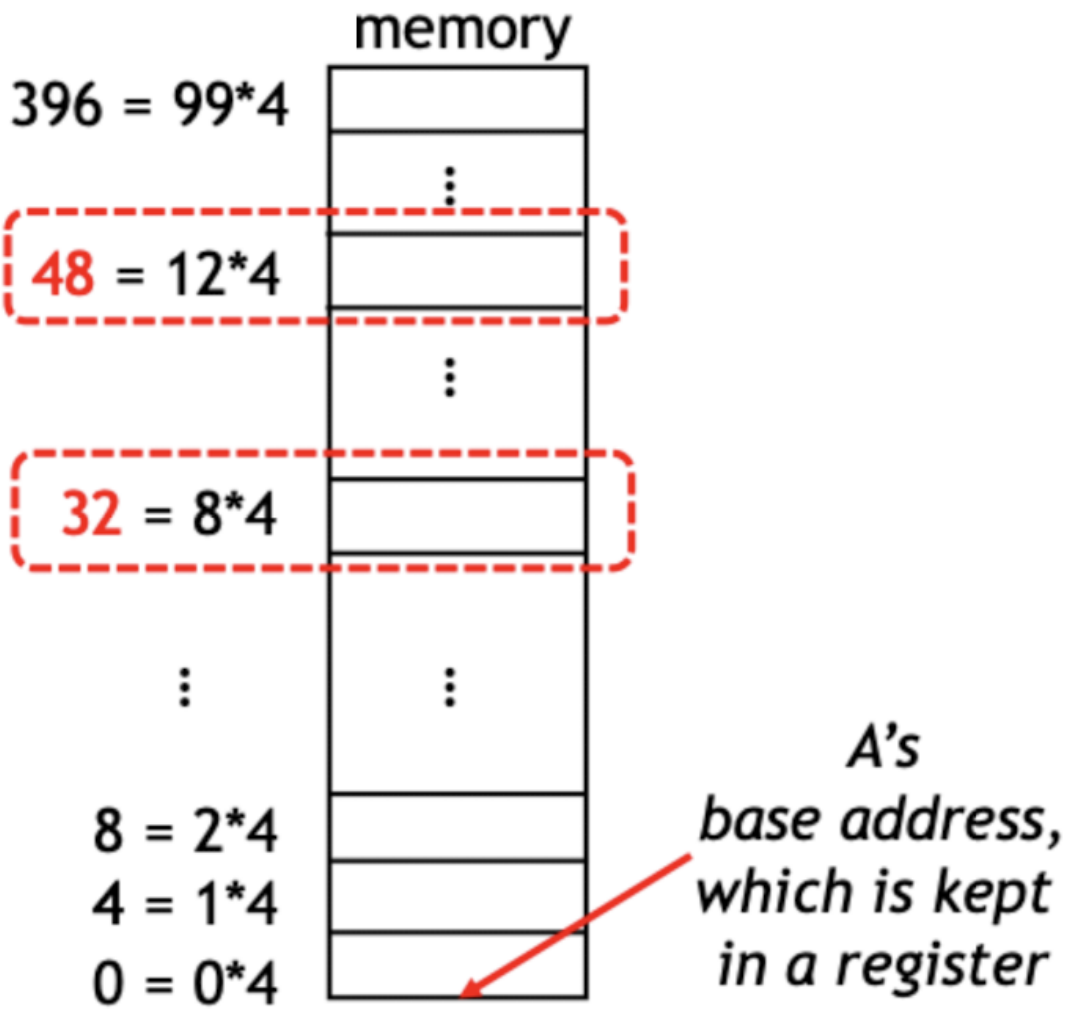
Review: Store Word Instruction

sw: 레지스터에서 메모리 위치로 데이터를 전송하는 명령어

- sw (store word)

    - sw register_to_be_stored memory_address_to_be_accessed (sw “저장된 레지스터” “접근 할 메모리 주소”)
    - memory_address_to_be_accessed: index(register that holds the base address of memory: 메모리의 기본 주소를 보유한 레지스터)
    - Here, index must be specified in bytes
- Example
    - Assumption 1: variable h is associated with register $s2
    - Assumption 2: the base address of the array A is in $s3
    - C code: A[12] = h + A[8]; is translated into (A[8]이 array라서 load로 데이터를 받아와야 함.)

```
lw $t0, 32($s3) # temporary reg $t0 gets A[8]
add $t0, $s2, $t0 # temporary reg $t0 gets h + A
[8]
sw $t0, 48($s3) # stores h + A[8] back into A[12]
```



Representing Numbers

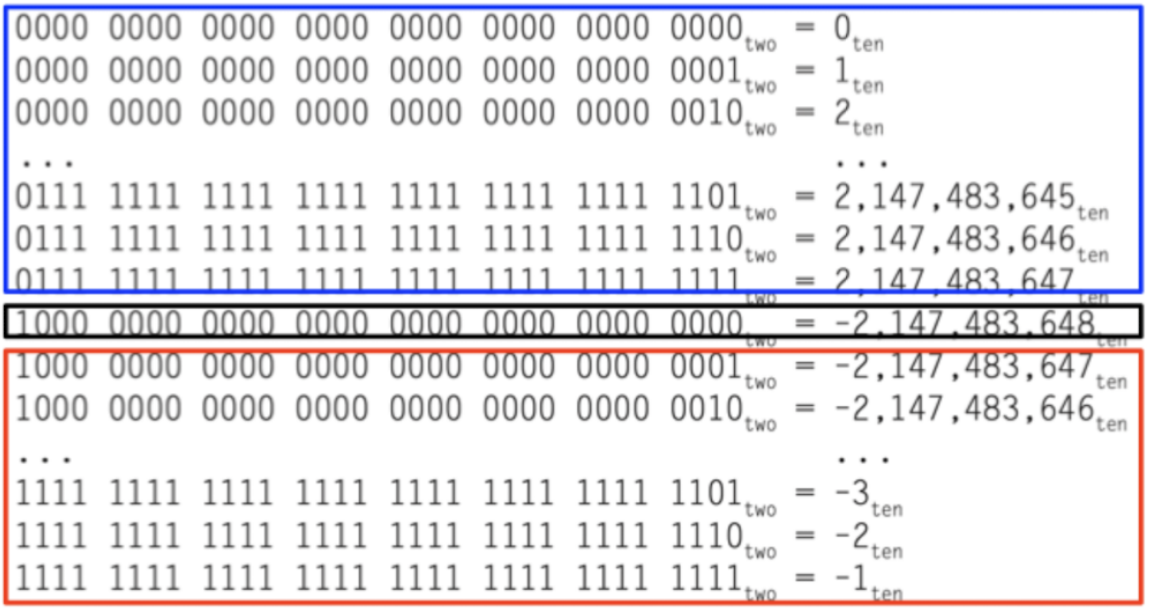
- Computer hardware (including MIPS processor) uses binary (base 2) numbers
  - We humans use decimal (base 10) numbers
  - 11 = (1 * 2^3) + (0 * 2^2) + (1 * 2^1) + (1 * 2^0) = 1011_2
  - 이진수는 2의 거듭제곱으로 표현되는데, 11을 이진수로 변환할 때는 가장 큰 2의 거듭제곱을 먼저 구하고, 그 다음에 큰 거듭제곱, 그 다음으로 큰 거듭제곱 순서로 값을 표현함.
- How does a MIPS processor understand/express a number?
  - Recall that MIPS is based on the unit a word (1word = 4bytes = 32bits)
  - MIPS numbers the 32 bits 0, 1, 2, 3, ... from right to left in a word

$11 = 1011_2$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

    - We refer to “bit 31” as most significant bit(최상위 비트), and “bit 0” as least significant bit(최하위 비트)
- How many numbers a MIPS word can represent?
    - Using 32 bits. we can express 2^32 = 4,294,967,296 different bit patterns (so different numbers)
    - We need to express both positive and negative numbers
    - How do we distinguish the positive from the negative?

Representing Numbers - Two’s Complement



- There have been different ways of preernting negative numbers
- Today’s processors pick two’s complement representation
    - This makes design of hardware simple
    - Leading 0 means that it’s a positive number
    - Leading 1 Means that it’s a negative number
- The positive half of the numbers are from 0 to 2,147,483,647 (2^31 - 1)
- 10000...0000 represents the most negative number - 2,147,483,648

- It is followed by a declining set of negative numbers from -2,147,483,647 to -1
- Negation (e.g., conversion between 2 and -2)
  - 2 = 0000 0000 0000 0000 0000 0000 0000 0010
  - (1) inverting the bits: 1111 1111 1111 1111 1111 1111 1111 1101
  - (2) adding one: 1111 1111 1111 1111 1111 1111 1111 1110 = -2

Representing MIPS Instructions

- MIPS instruction (add, sub, addi, lw, sw) are also information
  - So, instructions can be represented as numbers
- How can a MIPS instruction be represented as a binary number in a word?
- Example: add $t0, $s1, $s2

0	17	18	8	0	32
000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

  - 32bit is divided into segments, which is called fields (32비트는 세그먼트로 나뉘며, 이를 필드라고 한다.)
  - The first and last fields in combination tell that is an addition (첫번째와 마지막 필드의 포함은 이것이 덧셈임을 알려준다.) (add: 0 + 32)
  - The second field gives the number of the register that is first source of operands (두번째 필드는 피연산자의 첫번째 소스인 레지스터의 번호를 제공한다.) ($s1: 17)
  - The third field gives the number of the register that is the other source operands (세번째 필드는 다음 피연산자의 소스인 레지스터의 번호를 제공한다.) ($s2: 18)
  - The fourth field contains the number of the register that is the destination (네번째 필드에는 대상인 레지스터의 번호가 포함된다) ($t0: 8)
  - The fifth field is unused in this instruction (so, it is set to 0) (다섯번째 필드는 이 명령어에서 사용되지 않는다. 따라서 0으로 설정된다.)

Representing MIPS Instructions: R-Type

- An R-type instruction is comprised of six different fields (R 타입 명령어는 6가지 서로 다른 필드로 구성된다.)
  - op: basic operation of the instruction, called opcode (명령어의 기본연산, opcode라 한다.)
  - rs: the first register source operand (첫 번째 레지스터 소스 피연산자)
  - rt: the second register source operand (두 번째 레지스터 소스 피연산자)
  - rd: the register destination operand (레지스터 대상 피연산자)
  - shamt: shift amount (시프트 양) (this is used in shift instructions; otherwise, this field is filled with 0s: 시프트 명령어에 사용되며 그렇지 않으면 이 필드는 0으로 채워진다.)
  - funct: function, called function code, selects the specific variant of the operation in op (함수 코드라고 불리는 함수는 연산에서 특정 연산 변형을 선택한다.)
- Two representative R-type instructions we know

Instruction	Format	op	rs	rt	rd	shamt	funct
add	R	0	reg	reg	reg	0	32_{ten}
sub (subtract)	R	0	reg	reg	reg	0	34_{ten}

Representing MIPS instructions: Register Number

register	assembly name	Comment
r0	$zero	Always 0
r1	$at	Reserved for assembler
r2-r3	$v0-$v1	Stores results
r4-r7	$a0-$a3	Stores arguments
r8-r15	$t0-$t7	Temporaries, not saved
r16-r23	$s0-$s7	Contents saved for use later
r24-r25	$t8-$t9	More temporaries, not saved
r26-r27	$k0-$k1	Reserved by operating system
r28	$gp	Global pointer
r29	$sp	Stack pointer
r30	$fp	Frame pointer
r31	$ra	Return address

- A MIPS processor includes 32 registers
- Some are reversed for specific purposes
  - Register1 ($at) for assembler
  - Registers 26-27 ($k0-$k1) for operating system
- Many are used for specific purposes
  - Registers 2-7 for function calls
  - Registers 28-31 for program flow
- Programs use the other registers
  - $t0 to $t7 are mapped to 8 to 15
  - $s0 to $s7 are mapped to 16 to 23

    - $t8 and $t9 are mapped to 24 and 25
- Example: add $t0, $s1, $s2
    - rs = 17, rt = 18, rd = 8

Representing MIPS Instructions: I-Type

- R-type does not fit to other instructions (R 타입은 다른 명령어에 적합하지 않습니다.)
    - Data transfer instructions: 데이터 전송 명령어 (one register and a memory address: 레지스터 1개와 메모리 1개)
    - Immediate instructions: 즉시 명령어 (two registers and a constant: 레지스터 2개와 상수)
- An I-type instruction is comprised of four different fields (I형 명령어는 네 가지 필드로 구성된다.)
- Example: lw $t0, 32($s3)
    - op = instructions (lw = 35), rs = base address ($s3 = 19)
    - rt = destination register ($t0 = 8), constant/address = index(32)

Instruction	Format	op	rs	rt	rd	shamt	funct	address
add immediate	I	8_{ten}	reg	reg	n.a.	n.a.	n.a.	constant
lw (load word)	I	35_{ten}	reg	reg	n.a.	n.a.	n.a.	address
sw (store word)	I	43_{ten}	reg	reg	n.a.	n.a.	n.a.	address

Representing MIPS Instructions: Example

- Assumption: $t1 holds the base address of an integer array A, $s2 corresponds to h
- A[300] = h + A[300]; is compiled to

```
lw $t0, 1200($t1) # temporary reg $t0 gets A[300]
add $t0, $s2, $t0 # temporary reg $t0 gets h + A[300]
sw $t0, 1200($t1) # store h + A[300] back into A[300]
```

Op	rs	rt	rd	address/ shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

Representing MIPS Instructions: summary

Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add $s1,$s2,$s3
sub	R	0	18	19	17	0	34	sub $s1,$s2,$s3
addi	I	8	18	17	100			addi $s1,$s2,100
lw	I	35	18	17	100			lw $s1,100($s2)
sw	I	43	18	17	100			sw $s1,100($s2)
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer format

- R-type and I-type
- Each type has different fields with fixed sizes
- Opcode is unique for each instruction
- Registers as source/destination operands are specified their numbers