Computer Architecture (ENE1004)

▼ Lec 2

Lec 2: Instructions: Language of the Computer 1

Instruction Set

- To command a computer's hardware, you must speak its language
  - Instructions: the words of a computer's language
  - Instructions are hardware-specific: the language of a computer is different from those of other computers
  - Instruction set: the list of commands understood by a given computer
- Languages of computers might be diverse like human languages
  - Do we have to learn all kinds of computers' languages (instruction sets)?
  - No, in reality, computer languages (instruction sets) are quite similar
  - They are more like dialects than like independent languages
- The similarity of instruction sets comes from the following facts
  - Hardware technologies are based on similar underlying principles
  - There are a few basic operations the all hardware technologies must provide
- We pick and study the language of MIPS computer (MIPS instruction set)
  - ARM, Intel x86, MIPS, ...
  - Once you learn MIPS, it is easy to learn others
- CPU의 언어(CPU가 이해할 수 있는 언어)를 배워야 함. → 이 집합이 Instruction Set

- Instruction Set에는 여러 종류가 있음. 근데 CPU가 거의 비슷하기 때문에 언어들도 비슷함 (방언같은 개념).
- 우리는 굳이 복잡한 Arm을 배우지 않고 옛날에 나온 간단한 Mips를 배울 것임 (Mips의 CPU 구조).

MIPS Arithmetic Instructions (MIPS 산술 지침)

- An instruction set must include arithmetic operations (산술 연산)
- Additions(add) and subtraction(sub)
- An example of MIPS arithmetic instructions:

```
add a, b, c # sum of b and c into a
```

- This instructs a MIPS CPU to add two variables b and c and to put their sum in a
- Each MIPS arithmetic instruction performs only one operation (더하기를 한 번밖에 못함. 즉, 연산을 한 번밖에 못함.)
- Each MIPS arithmetic instruction always has exactly three variables (3개의 변수를 무조건 가져야 함.)
- An example of placing the sum of four variables b, c, d and e into variable a
    - In C, a = b + c + d + e;
    - add a, b, c # sum of b and c into a
    - add a, a, d # sum of a and d into a
    - add a, a, e # sum of a and e into a
    - This single task needs a sequence of three different arithmetic instructions
    - This words to the right of # (sharp symbol) are comments

Example: Compiling C Statements into MIPS Instructions (C 명령문을 MIPS 명령어로 컴파일 하기)

- A MIPS compiler translates a C program segment into a set of MIPS instructions

- Example 1: a C segment where five variables, a, b, c, d, and e are involved

```
a = b + c;                              add a, b, c;
                                           --->
d = a - e                               sub d, a, e;
# C program                         # MIPS instructions
```

- Example 2: a C segment where five variables, f, g, h, i and j are involved

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

      |
      |
        ▼

add t0, g, h # temporary variables t0 contains g + h
add t1, i, j # temporary variables t1 contains i + j
sub f, t0, t1 # f gets t0 - t1, which is (g + h) - (i + j)
# MIPS instructions
```

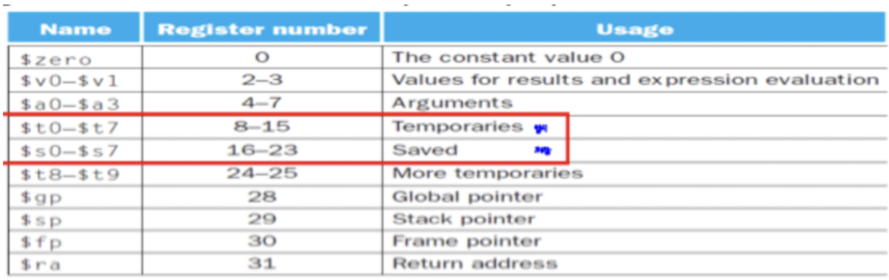
    - Recall that each MIPS instruction can be perform only one operation
    - So, temporary variables may be needed (t0, t1)
- Generally, assembly language code has more lines than high-level language code

Operands of MIPS Instructions: Registers

- The operand of arithmetic instructions are CPU registers (산술 명령의 피연산자는 CPU 레지스터임.)
    - In high-level languages, you can create and use a large number of variables
    - In MIPS instructions, you can use special locations within CPU, which are called registers

- # of registers in a MIPS CPU (so, # of operands in MIPS instructions) is limited
- 레지스터: CPU 내에 있으며, 메모리 같은 것임. 하지만 메모리는 CPU 밖에 있으므로 메모리는 아님.
- CPU는 연산만 하고 데이터 저장을 못하는데 데이터가 없으면 연산을 어떻게 할까?
- 잠시 저장하기 위해 아주 작은 메모리를 CPU 안에 만든 것이 레지스터임. MIPS에는 32 registers가 있음.
- 즉, 레지스터는 데이터 저장 용도가 아닌 계산을 위해 잠시 저장해서 읽고 쓸 용도임.

- Registers in the MIPS CPU
    - There are 32 registers
    - The size of each registers is 32 bits (a unit of 32 bits is called “word”)
    - word: 데이터 크기의 단위 (1word = 4bytes = 32bits)
- Representing registers from 0 to 31
    - Each register has its own name and specific purpose



Name	Register number	Usage
$zero	0	The constant value 0
$v0–$v1	2–3	Values for results and expression evaluation
$a0–$a3	4–7	Arguments
$t0–$t7	8–15	Temporaries
$s0–$s7	16–23	Saved
$t8–$t9	24–25	More temporaries
$gp	28	Global pointer
$sp	29	Stack pointer
$fp	30	Frame pointer
$ra	31	Return address

Operands of MIPS Instructions: Registers

Assuming that variables f, g, h, i, and j are assigned to registers $s0, $s1, $s2, $s3, and $s4

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

      |
      |
```

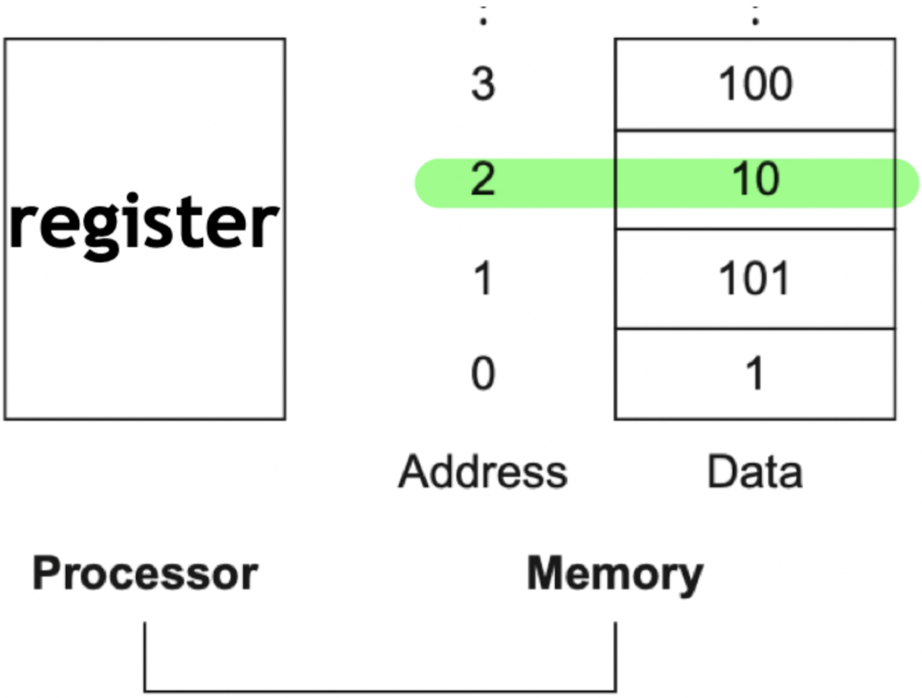
▼

```
add $t0, $s1, $s2 # register $t0 contains g + h
add $t1, $s3, $s4 # register $t1 contains i + j
sub $s0, $t0, $t1 # f gets $t0 - $t1, which is (g + h) - (i + j)
# Complete MIPS instructions
```

Operands of MIPS Instructions: Memory

- Then, how can MIPS CPU handle complex data structures using only 32 registers? (그렇다면 MIPS CPU는 어떻게 32개의 레지스터만을 사용해 복잡한 데이터 구조를 처리할 수 있을까?)
    - Arrays and structures can contain much more data elements than there are registers (배열과 구조체는 레지스터보다 훨씬 더 많은 데이터 요소를 포함할 수 있다.)
    - Data structures are kept in the memory (instead of registers) (데이터 구조체는 레지스터 대신 메모리에 보관된다.)
    - Recall thet the operands of arithmetic instructions must be registers (산술 명령의 피연산자는 레지스터여야 한다는 점을 기억하자.)
    - 메모리에서 가져와서 CPU(레지스터)에 올려놓고 연산함.
- So, MIPS must include data transfer instructions (load and stroe) (따라서 MIPS에는 데이터 전송 명령어 로드 및 저장이 포함되어야 한다.)
    - They instruct to transfer data between memory and registers (메모리와 레지스터 간에 데이터를 전송하도록 지시한다.)
    - Load: instruction to transfer data from a memory location to a register (메모리 위치에서 레지스터로 데이터를 전송하는 명령어)
    - Store: instruction to transfer data from a register to a memory location (레지스터에서 메모리 위치로 데이터를 전송하는 명령어)
- Specifying memory locations (메모리 위치 지정)
    - Memory is a large, single-dimensional array (메모리는 큰 단일 자원 배열이다.)
    - The unit is a word (32 bits) (단위는 word이다.)

- Address acting as index to that array, starting from 0 (주소는 해당 배열의 인덱스 역할을 하며, 0부터 시작한다.)
- The address of the 3rd data is 2 and its value is 10 (3번째 데이터의 주소는 2이고 그 값은 10이다.)



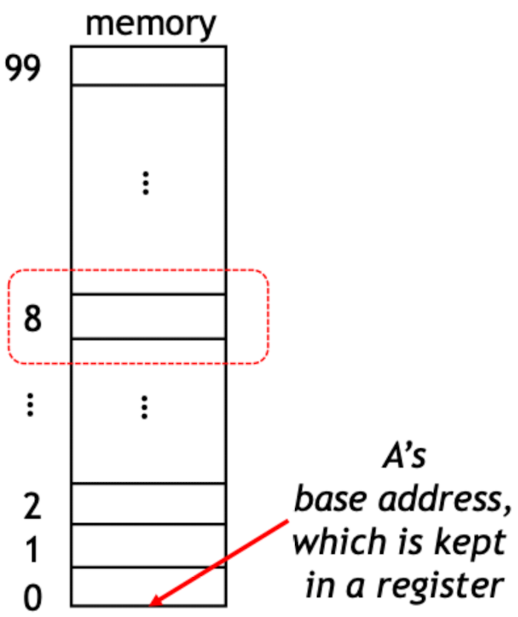
MIPS Data Transger Instructions: Load

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)
  - lw $s2, 2($s1)
- Example
  - Assumption 1: A is an array of 100 words
  - Assumption 2: starting(base) address of array A is in $s3
  - Assumption 3: Compiler associates variables g and h with $s1 and $s2

- C code: g = h + A[8]; is translated into (1개의 operation으로 계산할 수 있지만 A[8]이 array라서 load로 데이터를 받아와야 함.)

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

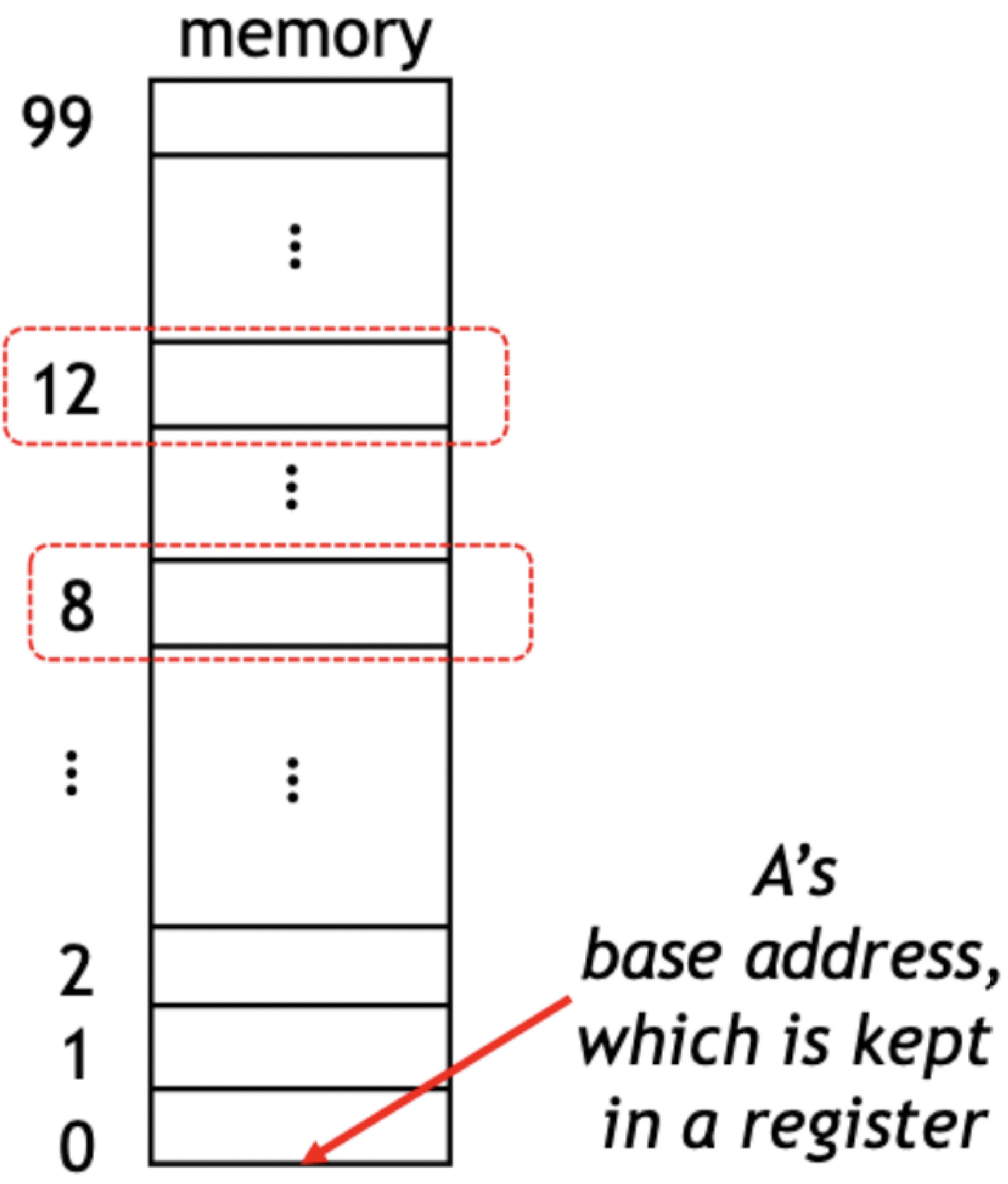


MIPS Data Transger Instructions: Store

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: address(register that holds the base address of memory: 메모리의 기본 주소를 보유한 레지스터)
    - sw $s2, 2($s1)
- 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 $t0, 8($s3)
add $t0, $s2, $t0
sw $t0, 12($s3)
```



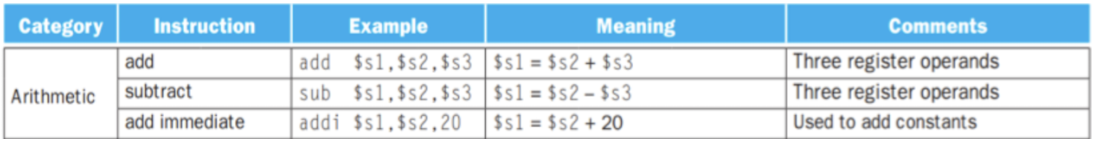
Operands of MIPS Instructions: Constant (상수)

- In many cases, a program use a constant in an operation (많은 경우, 프로그램에서 연산에 상수를 사용한다.)

    - Incrementing an index to point to the next element of an array (ex: 배열에 다음 요소를 가르키도록 인덱스를 증가시키는 경우)

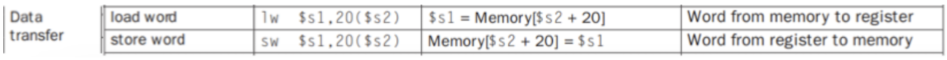
- MIPS includes immediate operations that use constant in arithmetic operations (MIPS에는 산술 연산에서 상수를 사용하는 즉시 연산이 포함된다.)
    - addi $s3, $s3, 4 # $s3 = $s3 + 4

- Compare and review the following three arithmetic instructions



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

- Review the following two data transfer instructions



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