Computer Architecture
(ENE1004)

Lec – 2: Instructions: Language of the Computer (Chapter 2) - 1

Instruction Set

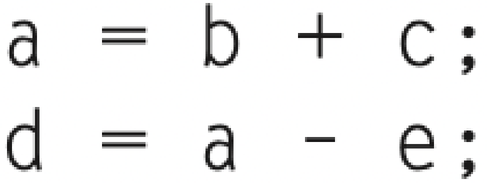
- To command a computer’s hardware, you must speak its language
  - Instructions: the words of a computer’s language
  - Instructions are hardware-specific: the language of a computer is different from those of other computers
  - Instruction set: the list of commands understood by a given computer
- 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 that 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

MIPS Arithmetic Instructions

- An instruction set must include is **arithmetic** operations
  - Addition (add) and subtraction (sub)
  - An example of MIPS arithmetic instructions: **add** *a*, *b*, *c*
  - This instructs a MIPS CPU to add the two variables *b* and *c* and to put their sum in *a*
- Each MIPS arithmetic instruction performs <u>only one operation</u>
- Each MIPS arithmetic instruction <u>always have exactly three variables</u>
- An example of placing the sum of four variables *b*, *c*, *d*, and *e* into variable *a*
  - **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
  - The words to the right of # (sharp symbol) are comments

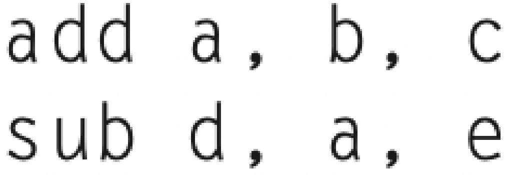
Example: Compiling C Statements into MIPS Instructions

- 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;
d = a - e;
```

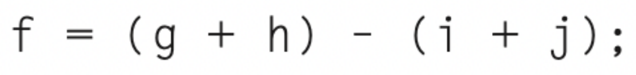
C program



```
add a, b, c
sub d, a, e
```

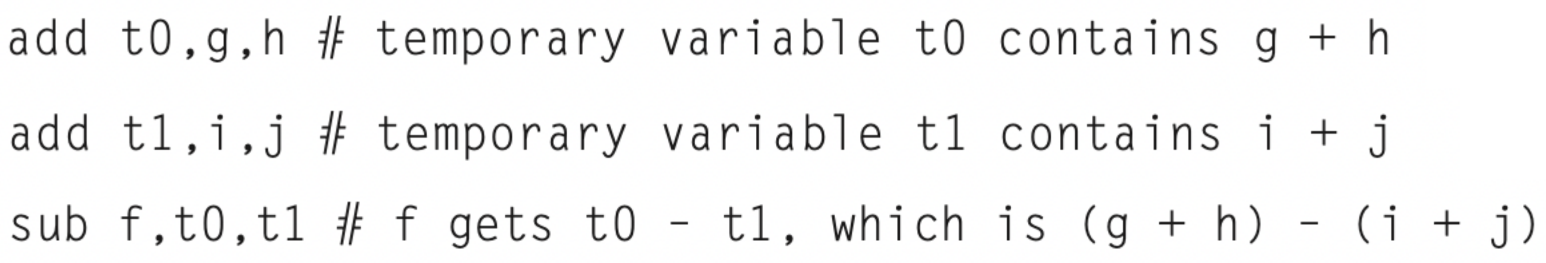
MIPS instructions

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



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

C program



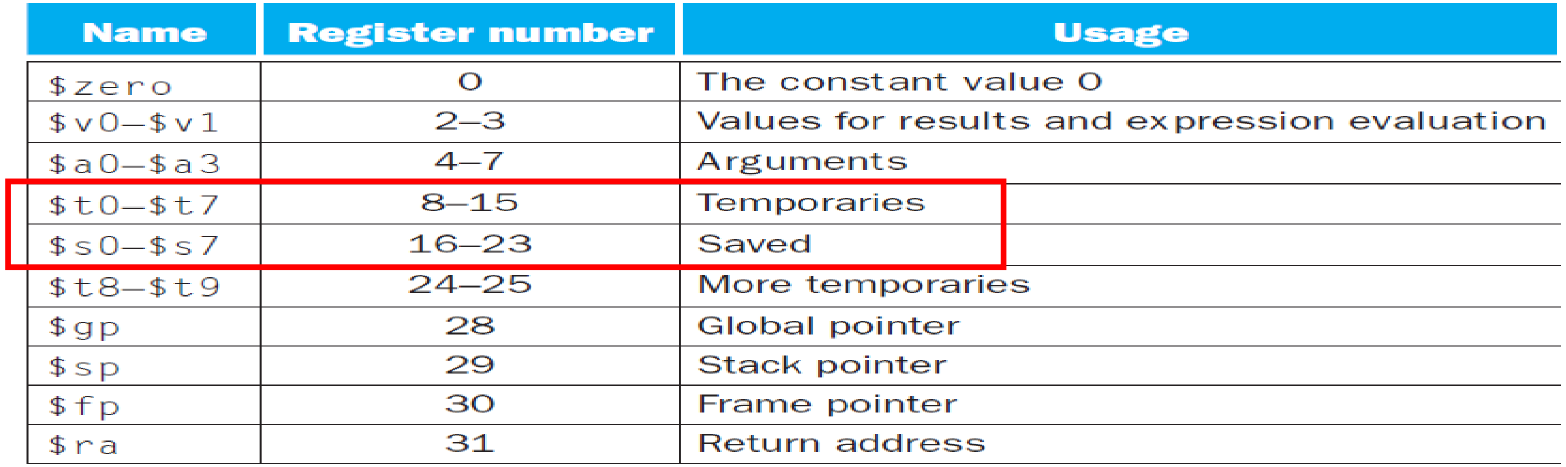
```
add t0,g,h # temporary variable t0 contains g + h
add t1,i,j # temporary variable t1 contains i + j
sub f,t0,t1 # f gets t0 - t1, which is (g + h) - (i + j)
```

MIPS instructions

  - Recall that only one operation can be performed for each MIPS instruction
  - 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

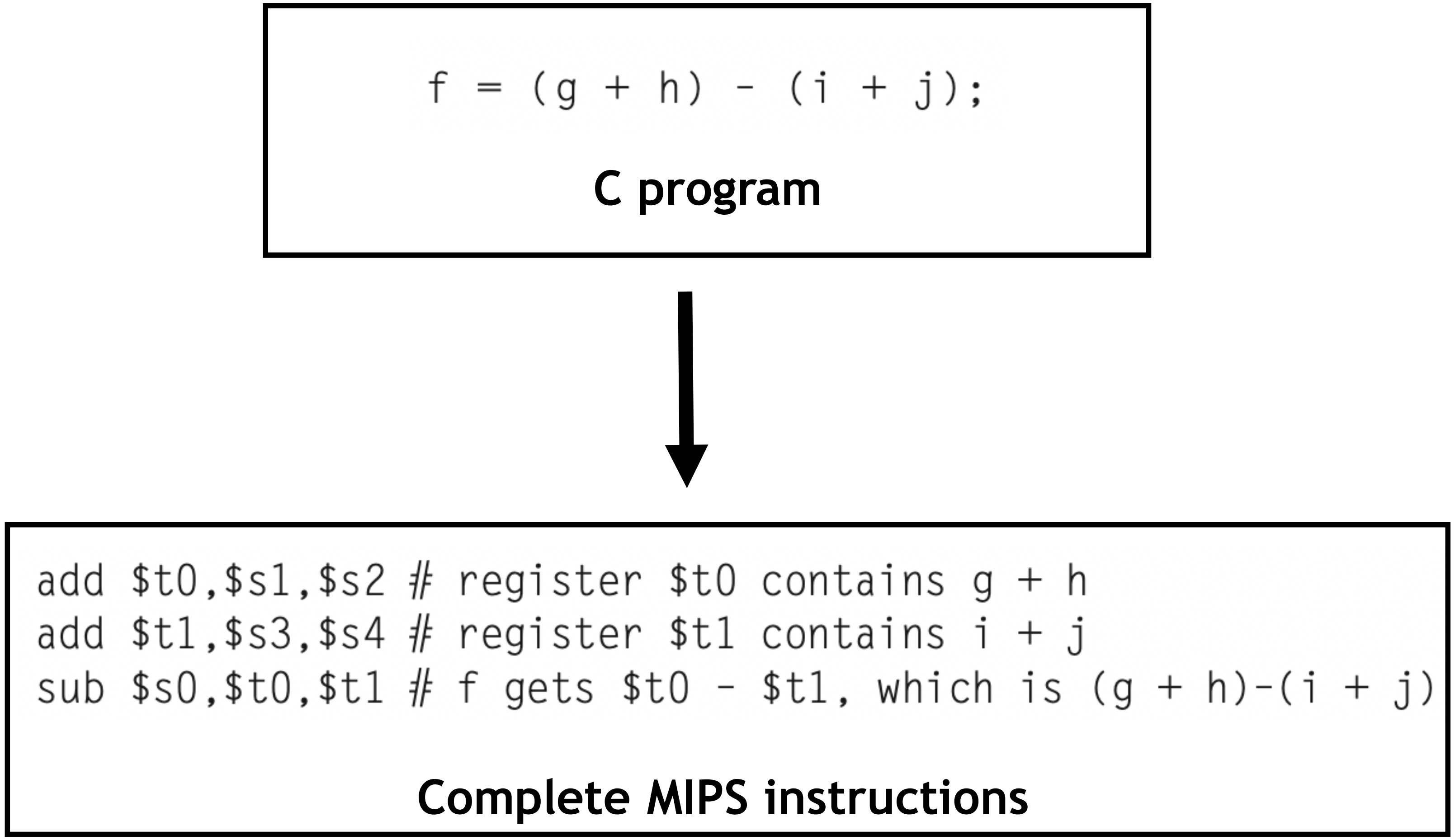
- The operands of arithmetic instructions are CPU registers
  - In high-level languages, you can create and use a large number of variables
  - In MIPS instructions, you can use special locations within CPU, which are called registers
  - # of registers in a MIPS CPU (so, # of operands in MIPS instructions) is limited
- Registers in the MIPS CPU
  - There are 32 registers
  - The size of each register is 32 bits (a unit of 32 bits is called “**word**”)
- 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);
```

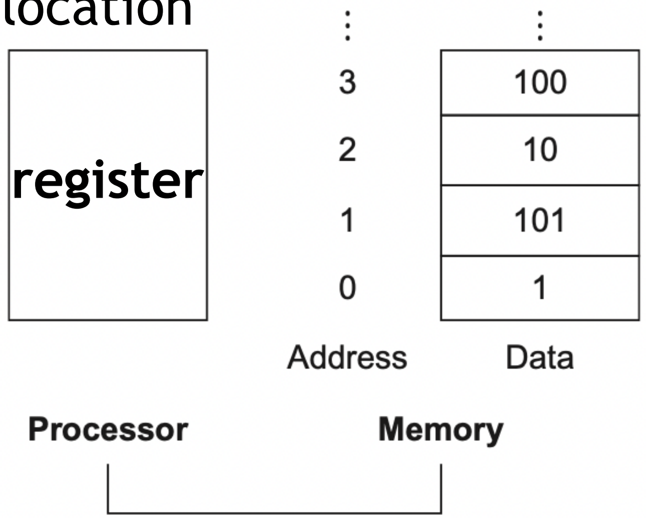
C program

```
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?
    - Arrays and structures can contain much more data elements than there are registers
    - Data structures are kept in the memory (instead of registers)
    - Recall that the operands of arithmetic instructions must be registers
- So, MIPS must include **data transfer** instructions (load and store)
    - They instruct to transfer data between memory and registers
    - Load: instruction to transfer data from a memory location to a register
    - Store: instruction to transfer data from a register to a memory location
- Specifying memory locations
    - Memory is a large, single-dimensional array
    - The unit is a word (32 bits)
    - Address acting as index to that array, staring from 0
    - The address of the 3rd data is 2 and its value is 10



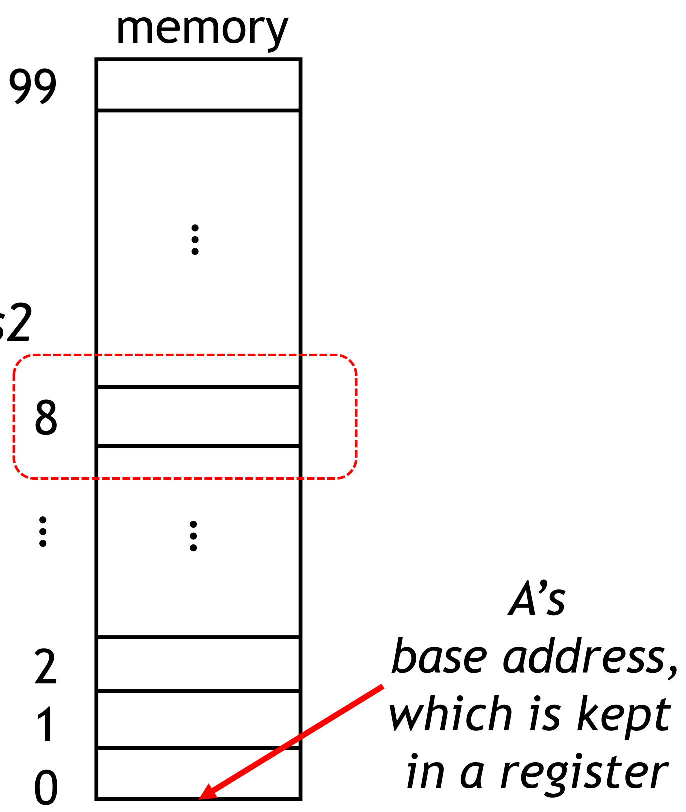
Address	Data
⋮	⋮
3	100
2	10
1	101
0	1

MIPS Data Transfer Instructions: Load

- **lw** (load word)
  - **lw** register_to_be_loaded memory_address_to_be_accessed
  - memory_address_to_be_accessed: index(register that holds the base address of memory)
  - **lw** $s2, 2($s1)
- Example
  - Assumption 1: A is an array of 100 words
  - Assumption 2: starting (base) address of array A is in *$s3*
  - Assumption 3: Compilers associated variables *g* and *h* with *$s1* and *$s2*
  - C code: **g = h + A[8];**

    is translated into
  - **lw $t0, 8($s3)   # temporary reg $t0 gets A[8]**

    add $s1, $s2, $t0   # g = h + A[8]



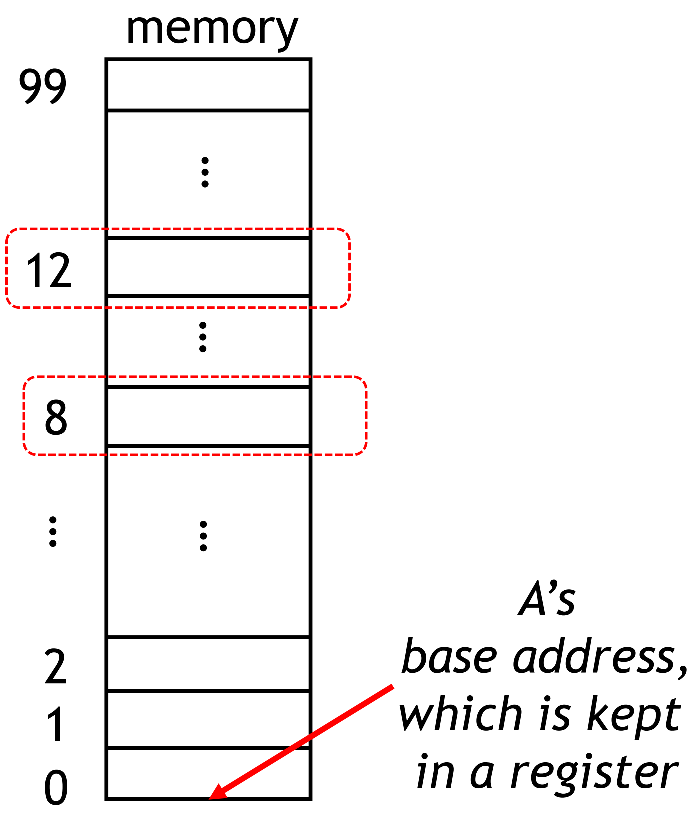
MIPS Data Transfer Instructions: Store

- **sw** (store word)
  - **sw** register_to_be_stored memory_address_to_be_accessed
  - memory_address_to_be_accessed: address(register that holds the base address of memory)
  - **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
  - **lw $t0, 8($s3) # temporary reg $t0 gets A[8]**

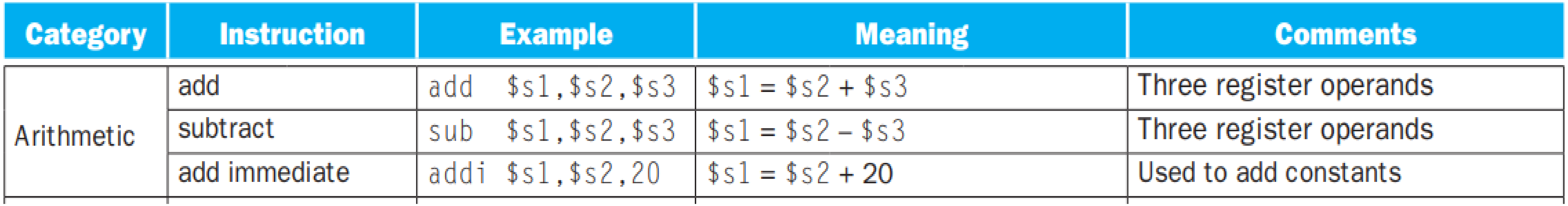
    add $t0, $s2, $t0 # temporary reg $t0 gets h + A[8]

    sw $t0, 12($s3) # stores h + A[8] back into A[12]



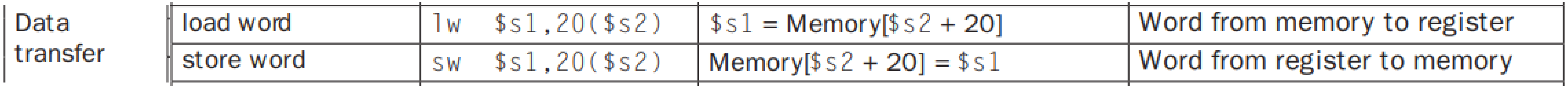
Operands of MIPS Instructions: Constant

- In many cases, a program use a constant in an operation
  - Incrementing an index to point to the next element of an array
- MIPS includes **immediate** operations that use constant in arithmetic operations
  - **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