CS47 - Lecture 07

Kaushik Patra
(kaushik.patra@sjsu.edu)

1

- Topics
  - Registers

Registers ...

2

Registers

- MIPS provides user accessible temporary storage called registers
  - Limited in number
  - Very fast to access
  - Each of the register is 32-bit (word) long
  - Group of registers are called register file

3

- Registers are the closest storage element to the processor. They resides inside processor, unlike main memory which exists on mother board external to the processor chip.
- These are the fastest storage that processor can access. Registers are designed and implemented with very fast circuit technology to make sure that the data access time is minimum. This is possible because very limited number of such storage has to be provided.

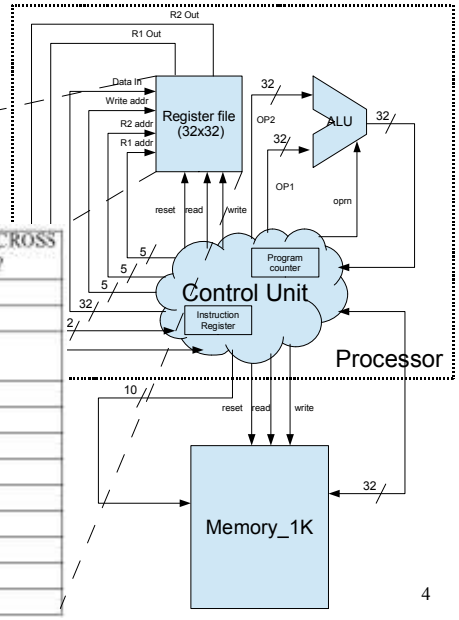
Register File

Special registers

Program Counter | Hi | Lo

General Purpose Registers

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	No



4

- There are 32 general purpose registers (GPR) in MIPS processor. Except for $zero, all other registers can be accessed and modified by user program. The register $zero acts as constant 0 value and is hardwired to retain value 0.

- Though all other 31 registers can be accessed and modified by, there are software convention that may limit the use of these temporary registers (discussed in next slide).

- There are three more registers, which are special purpose registers (SPR) in MIPS.

  - Program Counter (PC) – points to the latest instruction to be executed. This register can not be accessed or alter directly using data movement or math/logic operation. Branching & jumping operation can set the the value to PC indirectly.
  - High Reg (Hi) – To hold the results of mul / div operation (discussed later during multiplication and div operations)
  - Low Reg (Lo) – To hold the results of mul / div operation (discussed later during multiplication and div operations)

Register File Experiment

- Create a program exp5.asm
  - Include your cs47_macro.asm
  - Write a program using 'li' pseudo instruction to load all the registers from $zero to $ra with number 1 to 32. Use macro 'exit' to exit from program.
    - Example: li $zero, 1
  - Put a break point at 'exit'
  - Run the program and observe changes in registers.
    - $zero can not be written – it is always zero
- Create same program in exp6.asm but using the register numbers.
  - Example: li $0, 1
  - Observe that using the numbers is not that convenient to remember the purpose of the registers.

5

- Though all the 31 registers ($1 to $31) are accessible for user program to use and store temporary values – there are some conventions followed by software systems (user program, operating system, assembler, compiler, etc.)
- The register $at is used as assembler temporary – many pseudo instruction implementation uses this register. Therefore, it is not guaranteed that this register will not be altered without user program knowledge.
- Procedure calls (details will be discussed later) imposes many restrictions on the temporary storage usage. If the program is not using any procedure calls they are usually ok to use all the following registers without any adverse effect.
  - $v0-$v1 are to return values from procedures.
  - $a0-$a3 are to pass arguments to procedures.
  - $t0-$t9 values are not preserved across procedure calls.
  - $s0-$s7 values are preserved across procedure calls.

Data Movement in Registers

- No native support in MIPS processor except for Hi, Lo reg and to load upper 16 bits in registers
  - lui <reg> <Imm 16 upper bit value>
    - R[reg] = {Imm, 16'b0}
    - e.g. : lui $s1, 0x5f6a
- Assembler provides two pseudo instructions for integers.
  - Load immediate – 'li <reg>, <value>'
    - e.g. : li $t1, -19 # $t1 = -19
    - Limit for the value is 16-bit signed number range.
  - Move content of one reg to another
    - e.g. : move $t2, $t1 # $t2 = $t1
    - This is 32-bit value movement

6

- The registers $k0-$k1 is reserved for OS Kernel – therefore not guaranteed to be preserved. No user program should use this.
- $gp and $sp, are very special registers used for memory management / access (discussed later) – user programs are advised not to use these for temporary storage. They are to be used only for memory access.
- $fp and $ra are also very special registers used for procedure calls (discussed later). User program should not use them as temporary storage.
- The value limitation in 'li' pseudo instruction comes from the fact that internally it translate the 'li' into immediate (I) type native instruction. The value range can be embedded within I type instruction is bounded within 16-bit value range.
- Please note the 'move' command does not actually move the content from one register to another – it is just a copy operation.

Data Movement for Hi/Lo

- Since Hi & Lo can not be accessed directly MIPS provides four instructions to move to-n-fro information into and out of these two registers.
  - 'mfhi <reg>' : To copy from Hi to specified reg
  - 'mflo <reg>' : To copy from Lo to specified reg
  - 'mthi <reg>' : To copy to Hi from specified reg
  - 'mtlo <reg>' : To copy to Lo from specified reg
- Example:
  - mfhi $t0 : copy content of Hi to $t0
  - mtlo $s1: copy content of $s1 to Lo

7

- Any 32-bit integer multiplication operation gives 64-bit result. Therefore one 32-bit result destination register is not sufficient. The multiply operation ignores any 'rd' field specification in the machine code internally and place the results in these two Hi and Lo registers with higher bits at Hi and lower bits a Lo. Successive operations needs to retrieve the multiplication results from these two registers if needed.

- The integer division operation results in two integers – Quotient and remainder. These are two separate 32-bit numbers. Therefore, integer division also can not have just one result destination register. Similar to multiply operation division operation also ignores any 'rd' field specification in the machine code internally and place the results in these two Hi and Lo registers with quotient at Lo and remainder at Hi register.

Hi / Lo Data Move Experiment

- Write a program exp7.asm, which will ask two integer values for Hi and Lo and store it into Hi and Lo.
  - Include your cs47_macro.asm to access common macros.
- Extend the program to print values stored in Hi and Lo.
  - Create a macro in the program 'print_hi_lo' to do this.
- Extend the program to swap the values of Hi and Lo and then print the swapped values from Hi and Lo
- Observe the value changes in Hi, Lo in IDE

8

- Use the read_int macro to read in the integers – optionally use print_str macro to print appropriate messages.
- The print_hi_lo macro will take string locations for string “Hi”, “ = ”, “ , “ , “Lo” and print “Hi = <val> , Lo = <val>”. It uses print_str and print_reg_int macro from cs47_macro.asm.
- For swapping the content of Hi and Lo, read Hi and Lo into two temporary registers and then store them back in swapped order.
- Use break points in MARS to observe the data movement / copy in the registers at reach step of swaping.

'lui' Experiment

- Write a program exp8.asm, which store a 16-bit immediate value 0xa5a5 in $s1 and then use 'lui' to load 0x5a5a in the same register.
  - Include your cs47_macro.asm to access common macros.
  - Observe the value change in $s1 – the final content in the $s1 is not going to be 0x5a5aa5a5 but it is 0x5a5a0000.
- The 'lui' overrides the lower 16-bit with all 0s.
- Write a macro 'lwi' in cs47_macro.asm to load complete immediate 32-bit word into register.
  - Use 'ori' (OR immediate) (e.g. ori $t0, $t1, 0x5a5a => R[t0] = R[t1] | {16'b0, 0x5a5a}

- Write a macro 'lwi' (load word immediate) that takes three arguments as following:
  - Argument 1: Register name
  - Argument 2: Upper half word value
  - Argument 3: Lower half word value
  - Example: lwi $s2, 0xa5a5, 0x5a5a

```
# Macro: lwi
# Usage: lwi (<reg>, <upper imm>, <lower imm>)
.macro lwi ($reg, $ui, $li)
lui $reg, $ui
ori $reg, $reg, $li
.end_macro
```

CS47 - Lecture 07

Kaushik Patra
(kaushik.patra@sjsu.edu)

10

- Topics
  - Registers

