# CS4617 Computer Architecture

## Lecture 7a: Instruction Set Architectures (continued)

Dr J Vaughan

October 6, 2014

## Architectural expectations

- General-purpose registers, load/store architecture
- Addressing modes: displacement (address offset 12-16 bits), immediate (size 8-16 bits), register indirect
  Data sizes and types: 8, 16, 32, 64-bit integers; 64-bit IEEE 754 floating-point numbers
- Simple instructions: load, store, add, subtract, move register-register, shift
- Branches and jumps: compare equal, compare not equal, compare less, branch (PC-relative address at least 8 bits long), jump, call, return
- Instruction encoding fixed if performance more important, variable to minimize code size
- Minimal instruction set at least 16 GPRs, all addressing modes should apply to all data transfer instructions

## MIPS objectives

- Simple load-store instruction set
- Design for pipelining efficiency
- Fixed instruction set encoding
- Efficiency as a compiler target

## MIPS64 registers

- 32 $\times$ 64-bit GPRs (integer registers)
- Integer registers: R0, R1, ..., R31
- 32 $\times$ floating-point registers (FPRs)
- FPRs F0, F1, ..., F31 hold 32 single-precision (32-bit) values or 32 double-precision (64-bit) values
- When FPR holds single-precision number, the other half of the FPR is unused

## MIPS64 register operations

- Single and double-precision FP operations
- Instructions that operate on 2 single-precision numbers in a single 64-bit FPR
- Value of R0 is always zero
- Instructions that move between FPR and GPR
- Some special registers can be transferred to and from the GPRs. Example: FP status register

## MIPS64 Data types

- 8-bit bytes
- 16-bit half words
- 32-bit words
- 64-bit double words for integer data
- 32-bit FP single precision
- 64-bit FP double precision
- MIPS64 operations operate on 64-bit integers, 32-bit FP and 64-bit FP numbers
- Bytes, half words and words in 64-bit GPRs either have leading zeros or sign extension to fill out the 64 GPR bits. Once loaded, they are processed with 64-bit integer operations.

## MIPS addressing modes

- Immediate, 16-bit field
- Displacement, 16-bit field
- Register indirect achieved by putting 0 in displacement field
- Absolute 16-bit achieved by using R0 as the base register

## Memory addressing

- Memory is byte addressable, 64-bit address
- All memory accesses must be aligned
- Mode bit allows selection of Big Endian or Little Endian
- Load-store architecture means memory-register transfers are through loads or stores
- Memory accesses involving GPRs can be to a byte, half word, word or double word
- FPRs may be loaded/stored with single-precision or double-precision numbers

## MIPS64 Instruction Format

- 2 addressing modes can be encoded in the opcode
- 6-bit primary opcode
- 32-bit instruction
- 16-bit fields for displacement, immediate constants or PC-relative branch addresses

## Instruction types

- I-type: Opcode (6) / Rs (5) / Rt (5) / immediate (16)
- R-type: Opcode (6) / Rs (5) / Rt (5) / Rd (5) / shamt (5) / funct (6)
- J-type: Opcode (6) / Offset added to PC (26)

## MIPS Operations

- Four classes
  - Load and stores
  - ALU operations
  - Branches and jumps
  - Floating-point operations

## Notes on MIPS operations

- Any GPR or FPR may be loaded or stored
- Loading R0 has no effect
- Single-precision FP numbers use half a FP register
- Conversions between single and double precision must be done explicitly
- The FP format is IEEE 754

## Metalanguage 1

- C-like RTL with left arrow ($\leftarrow$) used for assignment
- *Mem* is main memory
- *Regs* denotes registers
- Mem [Regs[R1]] denotes "the contents of the memory location whose address is given by the contents of register R1"
- A subscript is appended to $\leftarrow$ to clarify the length of the data being transferred
- Thus $\leftarrow_n$ means "transfer an n-bit quantity"

## Metalanguage 2

- $x, y \leftarrow z$ means that z is transferred to x and y
- A subscript is used to show the selection of a bit from a field. Bits are labelled from the MSB starting with 0. The subscript can be a single digit ($Regs[R4]_0$ give the sign bit of R4) or a subrange ($Regs[R3]_{56..63}$ gives the LS byte of R3)
- The variable *Mem* , an array that signifies main memory, is indexed by a byte address and may transfer any number of bytes
- A superscript is used to replicate a field ($0^{48}$ is a field of zeros of length 48 bits)
- The symbol $\#\#$ is used to concatenate two fields and may appear on either side of a data transfer

## Example

- R8 and R10 are 64-bit registers
- $Regs[R10]_{32..63} \leftarrow_{32} (Mem[Regs[R8]]_0)^{24} \#\# Mem[Regs[R8]]$
- "the byte at the memory location with address given by the contents of register R8 is sign-extended to form a 32-bit quantity that is stored into the lower half of register R10."
- The upper half of R10 is unchanged.

## ALU instructions

- All ALU instructions are register-register
- Immediate forms of ALU instructions use a 16-bit sign-extended immediate
- LUI (load upper immediate) loads bits 32-47 of a register and sets the rest of the register to 0
- LUI allows a 32-bit constant to be built in two instructions or a data transfer using any constant 32-bit address one extra instruction.
- R0 can be used to implement load constant or move register to register
- LI: DADDIU R1, R0, #79 means $R1 \leftarrow 74$
- MOV: DADDU R1, R0, R7 means $R1 \leftarrow R7$

## MIPS control flow

- Compare instructions compare two registers to see if the first is less than the second
- *TRUE* : *destination register* $\leftarrow 1$
- *FALSE* : *destination register* $\leftarrow 0$
- Compare instructions set a register, so are called set-equal, set-not-equal, etc.
- Compare instructions have immediate forms

## Jump instructions

- Two ways to specify the destination address
- A link may be made or not
- Two jumps use a 26-bit offset shifted left by 2 bits and replace the lower 28 bits of the PC (of the next sequential instruction after the jump) to give the destination address
- Two jump instructions specify a register that contains the destination address
- Jump and link is used for procedure calls, placing the return address in R31

## Branches

- All branches are conditional
- The branch condition may test the source register for zero or nonzero.
- The source register may contain a data value or the result of a compare
- Can test if register contents are negative
- Can test for equality between registers
- Branch target address specified by 16-bit offset that is shifted left by two places and added to the PC
- To help with pipelining, many architectures have instructions that convert a simple branch into an arithmetic instruction
- MIPs uses conditional move on zero or not zero

## MIPS ALU instructions

| Example Instruction | Inst. Name | Meaning |
|---|---|---|
| DADDU R1,R2,R3 | Add unsigned | $Regs[R1] \leftarrow Regs[R2] + Regs[R3]$ |
| DADDIU R1,R2,#3 | Add immediate unsigned | $Regs[R1] \leftarrow Regs[R2] + 3$ |
| LUI R1,#42 | Load upper immediate | $Regs[R1] \leftarrow 0^{32}\#\#42\#\#0^{16}$ |
| DSLL R1,R2,#5 | Shift left logical | $Regs[R1] \leftarrow Regs[R2] << 5$ |
| SLT R1,R2,R3 | Set less than | $\mathit{if}\ (Regs[R2] < Regs[R3])$ $Regs[R1] \leftarrow 1$ $\mathit{else}\ Regs[R1] \leftarrow 0$ |

Table: MIPS arithmetic & logical instructions

## MIPS control flow instructions

| Example Instr | Instr Name | Meaning |
|---|---|---|
| J name | Jump | $PC_{36..63} \leftarrow name$ |
| JAL name | Jump and link | $Regs[R31] \leftarrow PC + 8;$ $PC_{36..63} \leftarrow name;$ $((PC+4) - 2^{27}) \leq name < ((PC+4) + 2^{27})$ |

Table: Typical MIPS control flow instructions