Lecture 4. Instructions

Functionality and usage

Yuri Panchul, 2014

Arithmetic

- addu rd, rs, rt
  - rd = rs + rt
- addiu rt, rs, imm - signed 16-bit value
  - rt = rs + sign_extend (imm)
- subu rd, rs, rt
- mul rd, rs, rt
- mult and div use special registers hi and lo
- Explanation about unused add, addi, etc

Synthetic forms of addu

- addu rd, rs
  - A mnemonic for addu rd, rd, rs
- addu rt, rs, imm

  A mnemonic for addiu rt, rs, imm
- addu rt, imm
- addiu rt, imm
  - A mnemonic for addiu rt, rt, imm

Synthetic forms of subu

- subu rd, rs
  - A mnemonic for subu rd, rd, rs
- subu rt, rs, imm

  A mnemonic for addiu rt, rs, -imm
- subu rt, imm
  - A mnemonic for addiu rt, rt, -imm
- negu rd, rs
  - A mnemonic for subu rd, $0, rs

Bitwise logical

- and rd, rs, rt
  - rd = rs & rt
- andi rt, rs, imm - unsigned 16
  - rd = rs & imm
- Similarly or, ori, xor, xori
- nor
  - rd = ~ (rs | rt)
- Note there is no nori

Synthetic logical

- move rd, rs
  - Register move
  - Mnemonics for: or rd, rs, $0

Shifts left

- sll rd, rt, shift
  - rd = rt << shift
- sllv rd, rt, rs
  - rd = rt << rs
- Assembler can convert sra mnemonic into srav

Arithmetic and logical shifts right

- Arithmetic shift - propagating the sign bit
  - sra rd, rt, shift
  - srav rd, rt, rs
  - Assembler can convert sra mnemonic into srav
  - Useful to implement signed division by 2^n
- Logical shift - bringing zeros into high bits
  - srl rd, rt, shift
  - srlv rd, rt, rs
  - Useful to implement unsigned division by 2^n

Load constant

- lui rt, imm - signed 16-bit value
  - Load upper immediate
  - rt = imm << 16
- li rd, constant
  - Synthetic instruction to load immediate constant
- la rd, address
  - Synthetic instruction to load address
  - Address expression may use labels

Different synthetic forms of li

- Form with unsigned 16-bit argument
  - imm >= 0 && imm < 0x10000 (65536)
  - ori rd, $0, imm
- Form with signed negative 16-bit argument
  - imm < 0 && imm >= -0xFFFF (-32768)
  - addiu rd, $0, imm
- Form for other 32-bit immediates
  - lui rd, hi16 (imm); ori rd, rd, lo16 (imm)

Set if...

- slt rd, rs, rt
  - if (signed (rs) < signed (rt)) rd = 1; else rd = 0
- slti rd, rs, imm - 16 bit signed immediate
  - if (signed (rs) < signed (imm)) rd = 1; else rd = 0
- sltu and sltiu - use unsigned comparison
- Note that in sltiu 16-bit immediate is still sign-extended to 32 bits, then compared using unsigned comparison

Synthetic set if...

- seq, sne
  - set if equal (==), set if not equal (!=)
- sge / sgeu
  - signed / unsigned set if greater than or equal (>=)
- sgt / sgtu
  - signed / unsigned set if greater than (>)
- sle / sleu
  - signed / unsigned set if less than or equal (<=)

Loads and stores

- lb rd, address
  - Load byte, sign-extended
  - Fill bits [31:8] in MIPS32, [63:8] in MIPS64 with bit 7
- lbu rd, address
  - Load byte, unsigned
  - Fill bits [31:8] in MIPS32, [63:8] in MIPS64 with 0
- lh, lhu
  - Similarly with 2-byte halfwords

Loads and stores - continue

- lw rd, address
  - Load 4-byte word
- sb rd, address
  - Store byte, signed and unsigned form is unnecessary
- sh rd, address
  - Store 2-byte halfword
- sw rd, address
  - Store 4-byte word

Branches, jumps and calls

- PC-relative branches (PC = program counter)
  - Short range - 256KB - use 16 bit word offset
  - Use condition
- Absolute addressed jumps with constant
  - Long range - 256MB - uses 26 bit word address
- Jump to register - full 32/64 bit in MIPS32/64
- Function calls (subroutines/procedures)
- All branches, jumps and calls use delay slots

Branches - beq, b, beqz

- beq rs, rt, label
  - if (rs == rt) goto label
  - Uses 16 bit word offset
  - Range +/- 128KB
- b label
  - A mnemonic for beq $0, $0, label
- beqz rs, label
  - A mnemonic for beq rs, $0, label

Branches - bgez, bge, bgeu

- bgez rs, label
  - Signed, if (rs >= 0) goto label
  - Signed comparison obviously
- bge rs, rt, label
  - Synthetic slt at, rs, rt; beq at, $0, label
  - Signed, if (rs >= rt) goto label
- bgeu rs, rt, label
  - Synthetic sltu at, rs, rt; beq at, $0, label
  - Unsigned, if (rs >= rt) goto label

Branches - bgtz, bgt, bgtu

- bgtz rs, label
  - Signed, if (rs > 0) goto label
  - Signed comparison
- bgt rs, rt, label
  - Synthetic slt at, rt, rs; bne at, $0, label
  - Signed, if (rs > rt) goto label
- bgtu rs, rt, label
  - Synthetic sltu at, rt, rs; bne at, $0, label
  - Unsigned, if (rs > rt) goto label

Compare bge and bgt

- bge rs, rt, label
  - slt at, rs, rt
  - beq at, $0, label
  - if ((rs < rt) == 0) goto label
- bgt rs, rt, label
  - slt at, rt, rs
  - bne at, $0, label
  - if ((rt < rs) == 1) goto label

Other branches

- Other true machine instruction branches
  - blez rs, label
    - Branch if less or equal to zero
  - bltz rs, label
    - Branch if less than zero
  - bne rs, rt, label
    - Branch if rs not equal to rt
- Other synthetic branches
  - ble, bleu, blt, bltu

Absolute jumps

- j label
  - Absolute addressed jumps with constant
  - Long range - 256MB
  - Uses 26 bit word address
  - Lowest 2 bits assumed to be 0
  - Highest 4 bits are kept as current PC
- jr rs
  - Jump to address in register
  - 32-bit in MIPS32,64 bit in MIPS64

Function calls - short range

- bgezal rs, label
  - if (rs >= 0) call function
  - Return address is unconditionally saved in ra ($31)
- bltzal rs, label

  if (rs < 0) call function

  Return address is unconditionally saved in ra ($31)
- bal label
  - Mnemonics for bgezal $0, label

Function calls - long range

- jal label
  - Return address is saved in ra ($31)
- jalr rd, rs
  - Store return address to rd, jump to the contents of rs
- jalr rs
  - Mnemonics for jalr ra, rs

Using stack for function calls

- Stack pointer - register sp ($29)
- Frame pointer - register fp ($30)
- The discussion how to use stack

MIPS64 - compatible with MIPS32

- In MIPS64 32-bit operands in 64-bit registers are treated as signed-extended
- Instructions like addiu simply continue to work
- Operand in load upper immediate (lui rt, imm) is sign-extended
  - Sign is copied in bits [63:32] of the target register
- Additional instructions for 64-bit operands

Additional MIPS64 instructions

- daddu, daddiu, dsubu, dmul
- dsll, dsllv, dsra, dsrav
- Constant shift >32 bit - dsll32, dsra32, dsrl32
- ld
- lwu
- Synthetic dla, dli

Instructions that do nothing

- Any instruction that targets $0
- nop - mnemonic for sll $0, $0, 0
  - Convenient because it encodes as 0x00000000
- ssnop - sll $0, $0, 1
  - Special treatment on superscalar processors
    - Guaranteed to spend at least 1 clock cycle to run

Instructions to discuss later

- syscall - causes a system call exception
- di and ei - disable and enable interrupts
- eret - return from exception
- cache - instruction to work with caches
- sync - memory access synchronizer

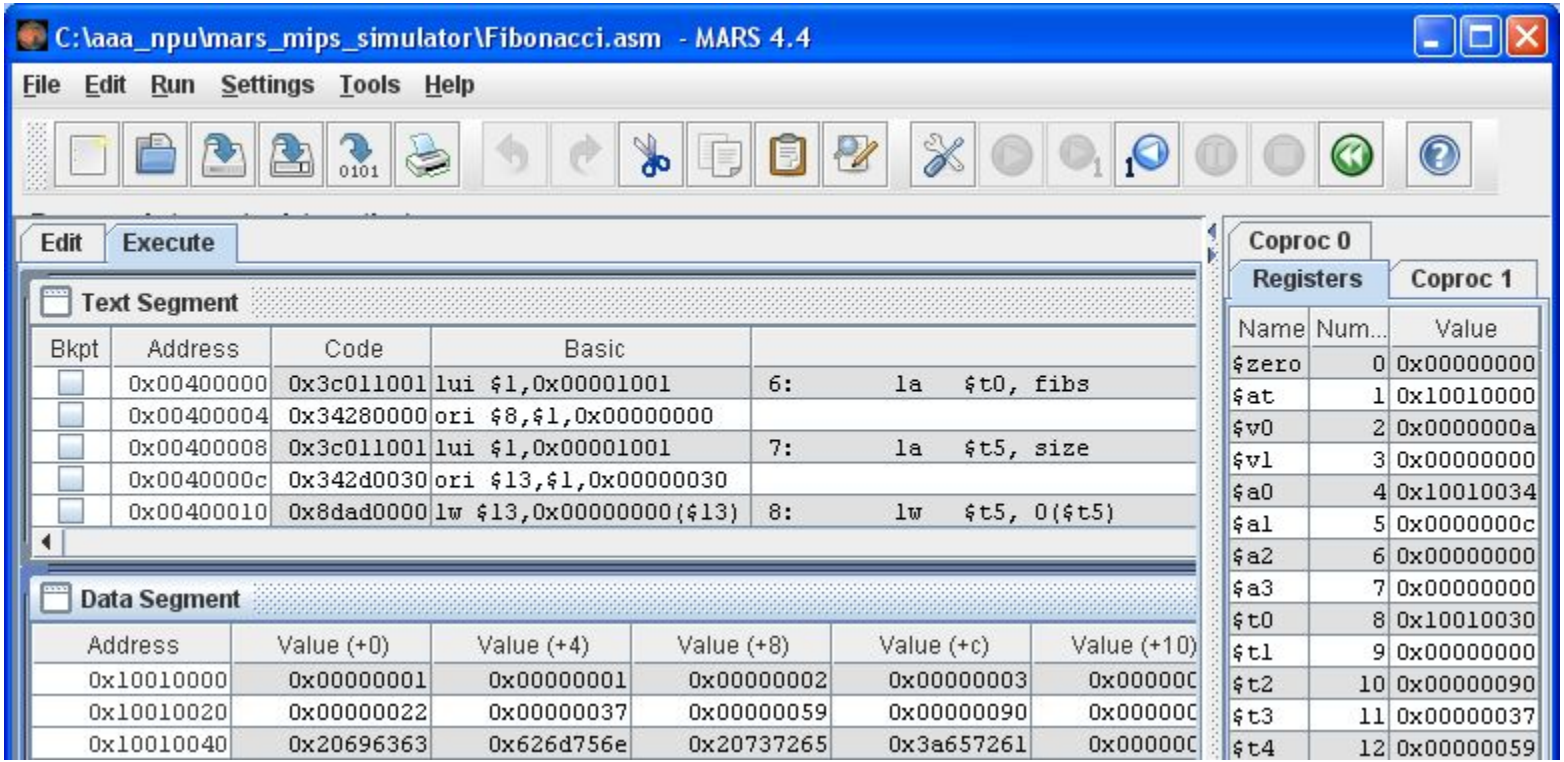
Instructions to discuss later - 2

- ll - load linked, for atomic access
- sc - store conditional, for atomic access
- mfc0 - move from coprocessor 0
- mtc0 - move to coprocessor 0
- ehb - execution hazard barrier, for coprocessor 0 access

Useful sometimes

- clo and clz - count leading ones and zeros
- ext and ins - bit manipulation
- rol and ror - rotations
- seb and seh - sign extension
- wsbh - byte swap
- MIPS64 additions to all the above
  - Except seb and seh, they don’t need MIPS64 version

How to practice - MARS simulator



Thank you!