CSI 333 – Lecture 16
MIPS Assembly Language (MAL): Part IV
(MAL and True Assembly Language (TAL))
MAL and TAL

- **MAL**: One level above True Assembly Language (TAL).
- TAL instructions are in one-to-one correspondence with MIPS machine instructions.
- A MAL instruction may be expanded into several instructions by the assembler (spim system).

**Simple Examples:**

<table>
<thead>
<tr>
<th>MAL instruction</th>
<th>TAL instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>move $7, $5</td>
<td>add $7, $0, $5</td>
</tr>
<tr>
<td>beqz $8, next</td>
<td>beq $8, $0, next</td>
</tr>
</tbody>
</table>
Multiplication and Division Operations

MAL version:

mul $8, $9, $10

To get the TAL equivalent, the following should be kept in mind.

- When two 32-bit quantities are multiplied, the result may be 64 bits long.
- MIPS has two special registers, called HI and LO, for 64-bit quantities. (These are different from the 32 general registers.)
- If product fits in 32 bits, then HI can be ignored.

TAL Equivalent:

mult $9, $10 #Result in HI, LO.
mflo $8
Multiplication and Division (continued)

**Notes:**

- The TAL opcode `mult` is different from the MAL opcode `mul`.
- The opcode `mflo` denotes “move from L0”.
- To check whether product fits in 32 bits:
  - If the value in L0 is non-negative, then HI must contain zero.
  - If the value in L0 is negative, then HI must contain `0xFFFFFFFF`. (This is because of 2’s complement representation: to be discussed later.)

**Division Operation:**

- TAL instruction computes the quotient in L0 and the remainder in HI.
The MAL instruction

\[
\text{div} \quad \$8, \$9, \$10
\]

is equivalent to the following TAL sequence:

\[
\text{div} \quad \$9, \$10 \quad \#\text{Quotient in LO.}
\]

\[
\text{mflo} \quad \$8
\]

Similarly, the MAL instruction

\[
\text{rem} \quad \$8, \$9, \$10
\]

is equivalent to the following TAL sequence:

\[
\text{div} \quad \$9, \$10 \quad \#\text{Remainder in HI.}
\]

\[
\text{mfhi} \quad \$8
\]

**Notes:**

- \text{mflo}, \text{mfhi} : Move from HI and LO (used in multiply/divide instructions as above).
- \text{mtlo}, \text{mthi} : Move to HI and LO.
TAL does not have `li` instruction.

The only TAL instructions that permit immediate operands are: `addi`, `andi`, `ori`, `xori` and `lui`.

Immediate operands can only be 16 bits long (because each instruction is only 32 bits long).

**Instruction** `addi`:

```
addi  Rd, Rs, I
```

**Effect:** The contents of register Rs and the immediate operand I are added and the result is placed in register Rd.

**Note:** There is no `subi` instruction in TAL. Instead, we must use `addi` with a negative immediate operand.
**Instruction andi:**

```plaintext
andi    Rd, Rs, I
```

**Effect:**

![Diagram showing the effect of andi instruction]

**Note:** Think of the immediate operand I as being extended to 32 bits by inserting 16 leading zeros. (This idea works for andi, ori and xori.)
### Immediate Mode Instructions (continued)

**Instruction `ori`:**

```plaintext
ori    Rd, Rs, I
```

**Effect:**

![Diagram showing the operation of the `ori` instruction]

**Instruction `xori`:**

```plaintext
xori   Rd, Rs, I
```

**Effect:** Similar to `ori`. 
**Instruction** `lui`: (Load Upper Immediate)

`lui` Rd, I

**Effect:**

![Diagram of Instruction and Effect]

- **Rd** (32 bits)
- **I** (16 bits)
- **Copy**
- **Old value lost**

**Exercise**

TAL does not have an `lli` (Load Lower Immediate) instruction. Synthesize the `lli` instruction using other TAL instructions.
Immediate Mode Instructions (continued)

Synthesizing MAL Instruction `li`:

MAL Segment:

```plaintext
li $5, 0xA1FBA0
```

Equivalent TAL Segment:

```plaintext
lui $5, 0x00A1
ori $5, $5, 0xFBA0
```
Synthesizing MAL Instruction la:

MAL segment:

```
la $5, val
```

Suppose the address of `val` (as determined by the assembler) is `0x01BCA010`.

Equivalent TAL Segment:

```
lui $5, 0x01BC
ori $5, $5, 0xA010
```
Arithmetic Ignoring Overflow

- Arithmetic operations may cause exception conditions such as overflow, underflow, division by zero.
- TAL provides instructions addu, subu, multu and divu that don’t cause any exception conditions to occur. (The suffix ’u’ denotes “unsigned”.)
- Useful in doing arithmetic on unsigned numbers. (No sign bit in that case.)

Digression – Exceptions in MIPS:

- MIPS normally includes two co-processors. Co-processor C0 is used to handle exceptions. (Co-processor C1 handles floating point instructions.)
- Exceptions cause interrupts.
Co-processor C0 has two important registers:

- **Cause Register** ($13 within C0): Indicates the reason for the interrupt.
- **Extended Program Counter or EPC** ($14 within C0): Gives the address of the instruction that caused the interrupt.

Exception handler: A software routine that uses registers $13 and $14 of C0 to process the interrupt.

Exception handler would use the instruction `mfc0` to copy a value from a register of C0 to one of the registers in the CPU.

**Example:**

```
  mfc0  $5, $14
```

The above instruction copies the contents of register $14 of C0 into register $5 of the CPU.
Unconditional Jump Instructions:

\[ j \quad \text{label} \]

**Effect:** Obvious.

\[ jr \quad $12 \quad \#\text{Jump register.} \]

**Effect:** Unconditional jump to the address stored in register $12.

\[ jal \quad \text{label} \quad \#\text{Jump and link.} \]

**Effect:** Store the address of the next instruction (i.e., the return address) in $31 and jump to label.

\[ jalr \quad $8, \quad $5 \quad \#\text{Jump and link register.} \]

**Effect:** Store the address of the next instruction (i.e., the return address) in $5 and jump to the address stored in $8.
Conditional Jump Instructions:

- Only bltz, bgez, blez, bgtz, beq and bne are provided in TAL.
- MAL instructions beqz and bnez can be synthesized using $0.

Synthesizing MAL Instruction blt:
Suppose we want to synthesize the following MAL instruction using TAL instructions:

```
blt $11, $12, iloop
```

A Wrong Try:

```
sub $1, $11, $12
bltz $1, iloop
```
**Difficulty:** Suppose $11$ contains the value $4 \times 10^9$ and $12$ contains the value $-4 \times 10^9$. (So, the branch to iLoop should not happen.)

- The TAL instruction
  
  \[
  \text{sub} \quad $1, \; $11, \; $12
  \]

  causes an overflow; so, the sub instruction may not work correctly.

- Replacing `sub` with `subu` won’t do. (The value in $12$ will be seen as a large positive number.)
A New TAL Instruction:

```
slt    Rd, Rs1, Rs2
```

Effect:

- Sets register Rd to 1 if the value in register Rs1 is (strictly) less than the value in register Rs2; otherwise, sets register Rd to 0.
- No exception conditions occur.
A Correct Implementation of \texttt{blt}:

MAL Statement:
\begin{verbatim}
blt $11, $12, iloop
\end{verbatim}

TAL Equivalent:
\begin{verbatim}
slt $1, $11, $12
bne $1, $0, iloop
\end{verbatim}

Recall: Register $1 is reserved by the assembler.
Other Forms of $slt$ Instruction

**Using an Immediate Operand:**

\[
\text{slti} \quad Rd, Rs1, I
\]

**Effect:** Sets register \( Rd \) to 1 if the value in register \( Rs1 \) is less than the immediate operand \( I \); otherwise, sets \( Rd \) to 0. (No exception conditions occur.)

**Example:**

\[
\text{slti} \quad $12, $10, 79
\]

**Note:** TAL also provides \( sltu \) and \( sltiu \), the “unsigned” versions of \( slt \) and \( slti \).
shift instructions in TAL

- MAL instructions of the form

  sll  $13, $12, 5
  srl  $13, $12, 6
  sra  $13, $12, 9

  where the shift amount is specified as an immediate operand, are actually TAL instructions.

- TAL also provides instructions of the following form:

  sllv $13, $12, $8
  srlv $13, $12, $4
  srav $13, $12, $3

  Here, the shift amount is specified as the contents of the third register specified in the instruction. (The suffix ‘v’ denotes “variable shift”.)