Move Instructions

Move instructions move data between registers.

Unconditional move instructions

mov instruction has two formats:

```
mov   s_reg, d_reg          mov   $T0, $T1
mov   value, d_reg          mov   2, $T0
```

T0 = 1 then mov $T0, $T1 results $T1 = 1

Conditional move instructions

`cmovXY` can have two or three operands. The first operand must be a register. The instruction is only executed if the value of the first operand satisfies the condition specified by the instruction. `XY` can be as below:

`eq` The instruction is executed if the first operand is equal to zero.
`ne` The instruction is executed if the first operand is not equal to zero.
`lt` The instruction is executed if the first operand is less than zero.
`le` The instruction is executed if the first operand is less than or equal to zero.
`gt` The instruction is executed if the first operand is greater than zero.
`ge` The instruction is executed if the first operand is greater than or equal to zero.

The format of the conditional move instruction with three operands is as below:

```
cmovXY   s_reg1, s_reg2, d_reg cmovXY  $T0, $T1, $T2
cmovXY   s_reg,  value,  d_reg cmovXY  $T0, 2, $T1
```

`Cmovne` $T0, $T1, $T2

- Where T0 = 0x02 and T1 = 3 executed T2 = 3
- Where T0 = 0x03 and T1 = 3 not executed, T2 unchanged

`Cmovle` $T0, 3, $T1

- Where T0 = 0xf…f executed T1 = 3
- Where T0 = 0x4 not executed T1 unchanged
- Where T0 = 0 executed T1 = 3

The format of the conditional move instruction with two operands is as follow:

```
cmovXY   d_reg/s_reg1, s_reg2      cmovXY   $T0, $T1
cmovXY   d_reg/s_reg, value        cmovXY   $T0, 2
```

- `The destination register is the first one.`

`Cmovge` $T0, $T1

- where T0 = 0x02 and T1 = 2 executed T0 = 2
- where T0 = 0xf…f and T1 = 2 not executed, T0 unchanged

`Cmovlt` $T0, 3

- where T0 = 0xf…f executed T0 = 3
- where T0 = 0x4 not executed T0 = 0x4

More Arithmetic Instructions

`clr` instruction sets the value of a register to 0, e.g. `clr $T0`
Only one operand
Can be done with mov # mov 0, $T0

`absq` instruction calculates the absolute value of a quadword. For example:

```
absq -0x123, $T0       $T0 = |-0x123|
```

T0 = 123

```
absq $T0           $T0 = |$T0|
If $T0 is -123, T0 will be 123

absq $T0, $T1      $T1 = |$T0|
```

If T0 is –123, T1 will be 123, T0 unchanged

`negq` instruction negates the contents of an operand. For example:

```
negq 0x123, $T0    $T0 = -0x123
negq $T0           $T0 = -$T0
negq $T0, $T1      $T1 = -$T0
```

Example: What are the contents of the registers after the execution of the program below? The content of each of the registers should be written as a quadword.

```
.
.
.
code{
     mov       10, $T0
     mov       -2, $T1
     mov       $T0, $T2
     cmovle    $T0, 2
     cmovgt    $T2, $T1, $T3
     cmovne    $T1, -5
     }
.
.
.
```

Answer:

```
     mov       10, $T0         # T0 = 10   T0 = 0x000000000000000a
     mov       -2, $T1         # T1 = -2   T1 = 0xfffffffffffffffe
     mov       $T0, $T2        # T2 = 10   T2 = 0x000000000000000a
     cmovle    $T0, 2          # not executed
     cmovgt    $T2, $T1, $T3   # T3 = -2   T3 = 0xfffffffffffffffe
     cmovne    $T1, -5         # T1 = -5   T1 = 0xfffffffffffffffb
```

Another example

There are no integer division instructions in Alpha.
You may implement integer division using addition and subtraction statements.

```
void main () {
  int quotient, remainder, divider=3, dividend=10;
  quotient = 0;
  remainder = dividend;
  while (remainder >= divider) {
    quotient++;
    remainder = remainder - divider;
  }
}
```

Convert the above C program to an assembly program. It should be assumed that (a) each variable is a quadword, (b) `dividend` is stored in $S0, (c) `divider` is stored in $S1, (d) `quotient` is in $S2 and (e) `remainder` is in $S3. The converted program should have the same control structure as the C program.

```
.
.
.
code{
        mov     10, $S0       // $S0 is dividend
        mov     3, $S1        // $S1 is divider
        mov     0, $S2        // $S2 is quotient
        mov     $S0, $S3      // $S3 is remainder
check:  cmple   $S1, $S3, $T0 //check whether
divider <= remainder
        beq     $T0, stop
        addq    $S2, 1
        subq    $S3, $S1
        br      check
    }
.
.
```