Foundations of Programming Languages

2OPM Assembly (2/3): Advanced Operations

Prof. Dr. Christoph Reichenbach

Fachbereich 12 / Institut für Informatik

22. Oktober 2014

Looping: Jumps and Conditionals

Compute factorial of `$t1` in `$t0`:

```
Assembly
li      $t0, 1
mul     $t0, $t1
subi    $t1, 1
bnez    $t1, -21     # If $t1 <> 0, then ...
```

Looping: Jumps and Conditionals

Compute factorial of `$t1` in `$t0`:

Assembly

```
li      $t0, 1
mul     $t0, $t1     # ... jump back here
subi    $t1, 1
bnez    $t1, -21     # If $t1 <> 0, then ...
```

- **bnez** `$r, n`:
  if $\$r \neq 0$, then $\$pc := \$pc + n$
  (conditional jump)

Looping: Jumps and Conditionals

Compute factorial of `$t1` in `$t0`:

Assembly
`li $t0, 1`
`mul $t0, $t1 # ... jump back here`
`subi $t1, 1`
`bnez $t1, -21 # If $t1 <> 0, then ...`

- **bnez** `$r, n`:
  if `$r` $\neq 0$, then `$pc` := `$pc` + n
  (conditional jump)
- Meaning of -21: Number of bytes in memory to jump

Looping: Jumps and Conditionals

Compute factorial of `$t1` in `$t0`:

Machine code	Assembly		
`49 ba 01 00 00 00 00 00 00 00`	**li**	`$t0, 1`	
`4d 0f af d3`	**mul**	`$t0, $t1`	# ... jump back here
`49 81 eb 01 00 00 00`	**subi**	`$t1, 1`	
`49 83 c3 00 0f 85 eb ff ff ff`	**bnez**	`$t1, -21`	# If $t1 <> 0, then ...

- **bnez** `$r, n`:
  if `$r` $\neq 0$, then $pc := $pc + n
  (conditional jump)
- Meaning of -21: Number of bytes in memory to jump

Looping: Jumps and Conditionals

Compute factorial of `$t1` in `$t0`:

Machine code	Assembly		
`49 ba 01 00 00 00 00 00 00 00`	**li**	`$t0, 1`	
`4d 0f af d3`	**mul**	`$t0, $t1`	# ... jump back here
`49 81 eb 01 00 00 00`	**subi**	`$t1, 1`	
`49 83 c3 00 0f 85 eb ff ff ff`	**bnez**	`$t1, -21`	# If $t1 <> 0, then ...

- **bnez** `$r, n`:
  if `$r` $\neq 0$, then `$pc` := `$pc` + n
  (conditional jump)
- Meaning of -21: Number of bytes in memory to jump

Jump distances like -21 hard to compute by hand

Labels

```
        li     $t0, 0x1
loop:
        mul    $t0, $t1
        subi   $t1, 0x1
        bnez   $t1, loop
```

- Label ‘loop’ could be any name

Labels

```
        li    $t0, 0x1
loop:
        mul   $t0, $t1
        subi  $t1, 0x1
        bnez  $t1, loop
```

- Label 'loop' could be any name
- Labels have no associated machine code
- Assembler replaces them with numbers

Labels

```
        li    $t0, 0x1
loop:
        mul   $t0, $t1
        subi  $t1, 0x1
        bnez  $t1, loop
```

- Label ‘`loop`’ could be any name
- Labels have no associated machine code
- Assembler replaces them with numbers

Exactly the same program as on the last slide

Jumps and Branches

- Unconditional jump: `j` ℓ

Jumps and Branches

- Unconditional jump: **j** ℓ
- Condition: compare two registers
  - **ble** `$r0, $r1,` ℓ # Jump if `$r0` <= `$r1`
  - Analogous: **bgt** ($>$), **bge** ($\geq$), **beq** ($=$), **bne** ($\neq$), ...

Jumps and Branches

- Unconditional jump: **j** ℓ
- Condition: compare two registers
  - **ble** `$r0, $r1,` ℓ # Jump if `$r0` <= `$r1`
  - Analogous: **bgt** ($>$), **bge** ($\geq$), **beq** ($=$), **bne** ($\neq$), ...
- Condition: compare register against 0:
  - **beqz** `$r0,` ℓ # Jump if `$r0` = 0
  - Analogous: **bnez**, **bgtz**, **blez**, ...

Jumps and Branches

- Unconditional jump: **`j`** ℓ
- Condition: compare two registers
  - **`ble`** `$r0, $r1,` ℓ `# Jump if $r0 <= $r1`
  - Analogous: **`bgt`** ($>$), **`bge`** ($\geq$), **`beq`** ($=$), **`bne`** ($\neq$), ...
- Condition: compare register against 0:
  - **`beqz`** `$r0,` ℓ `# Jump if $r0 = 0`
  - Analogous: **`bnez`**, **`bgtz`**, **`blez`**, ...

Operational Semantics for jumps is quite complex, won't be covered here

Memory access

Load: **ld** `$r0, d($r1)`
Store: **sd** `$r0, d($r1)`

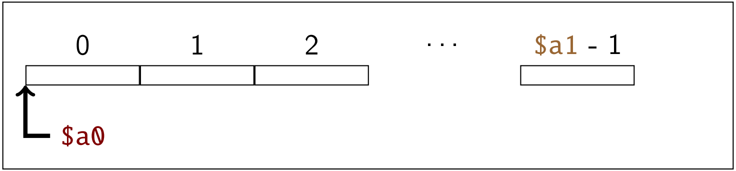
- `d`: 32 bit (signed) *displacement*
- Operation accesses memory address `d` + `$r1`
- Reads or writes 64 bit string to/from `$r0`

Finding the largest element

- Starting at address $a0:
- Search for largest 64-bit (8-byte) element in $a1 entries
- Store result in $v0

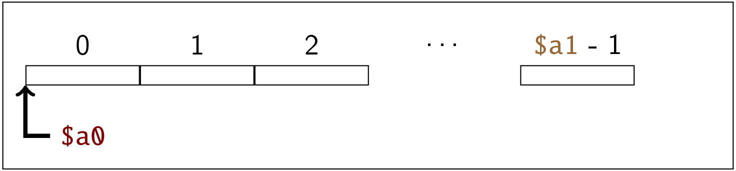
```
⇒       li      $v0, 0            # initial result
        move    $t0, $a1
        slli    $t0, 3            # x « 3 ≡ x × 2^3 = 8x
        add     $t0, $a0          # last addr. plus 64-bit word
loop:
        beq     $t0, $a0, end
        ld      $t1, 0($a0)       # load from memory
        addi    $a0, 8            # advance to next element
        blt     $t1, $v0, loop    # found bigger element?
        move    $v0, $t1
        j       loop
  end:
```

Finding the largest element



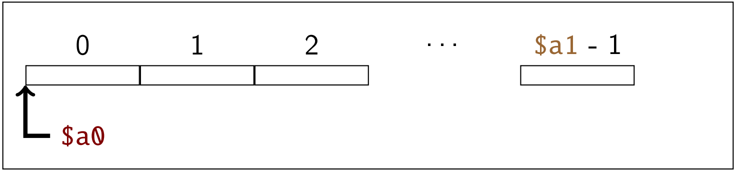

```
            li      $v0, 0             # initial result
⇒           move    $t0, $a1
            slli    $t0, 3             # x « 3 ≡ x × 2^3 = 8x
            add     $t0, $a0           # last addr. plus 64-bit word
loop:
            beq     $t0, $a0, end
            ld      $t1, 0($a0)        # load from memory
            addi    $a0, 8             # advance to next element
            blt     $t1, $v0, loop     # found bigger element?
            move    $v0, $t1
            j       loop
  end:
```

Finding the largest element



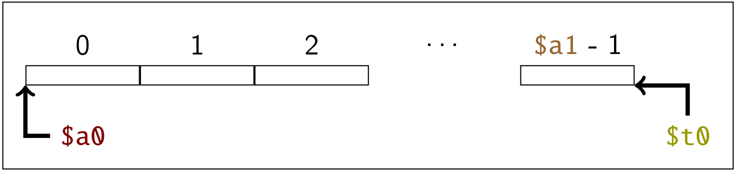

```
        li     $v0, 0            # initial result
        move   $t0, $a1
⇒       slli   $t0, 3            # x « 3 ≡ x × 2^3 = 8x
        add    $t0, $a0          # last addr. plus 64-bit word
loop:
        beq    $t0, $a0, end
        ld     $t1, 0($a0)       # load from memory
        addi   $a0, 8            # advance to next element
        blt    $t1, $v0, loop    # found bigger element?
        move   $v0, $t1
        j      loop
  end:
```

Finding the largest element



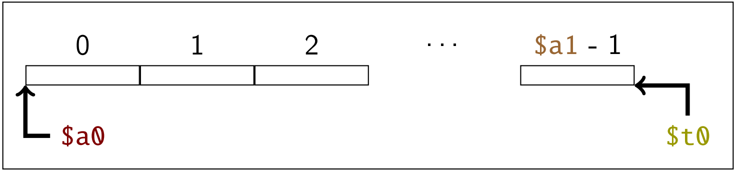

```
        li      $v0, 0          # initial result
        move    $t0, $a1
        slli    $t0, 3          # x « 3 ≡ x × 2^3 = 8x
⇒       add     $t0, $a0        # last addr. plus 64-bit word
loop:
        beq     $t0, $a0, end
        ld      $t1, 0($a0)     # load from memory
        addi    $a0, 8          # advance to next element
        blt     $t1, $v0, loop  # found bigger element?
        move    $v0, $t1
        j       loop
  end:
```

Finding the largest element



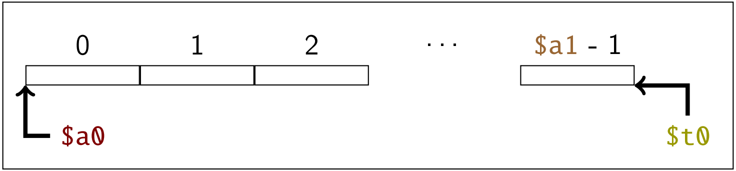

```
        li      $v0, 0              # initial result
        move    $t0, $a1
        slli    $t0, 3              # x « 3 ≡ x × 2^3 = 8x
        add     $t0, $a0            # last addr. plus 64-bit word
loop:
⇒       beq     $t0, $a0, end
        ld      $t1, 0($a0)         # load from memory
        addi    $a0, 8              # advance to next element
        blt     $t1, $v0, loop      # found bigger element?
        move    $v0, $t1
        j       loop
  end:
```

Finding the largest element



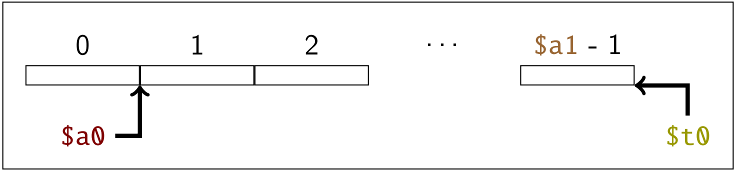

```
        li      $v0, 0              # initial result
        move    $t0, $a1
        slli    $t0, 3              # x « 3 ≡ x × 2^3 = 8x
        add     $t0, $a0            # last addr. plus 64-bit word
loop:
        beq     $t0, $a0, end
⇒       ld      $t1, 0($a0)         # load from memory
        addi    $a0, 8              # advance to next element
        blt     $t1, $v0, loop      # found bigger element?
        move    $v0, $t1
        j       loop
  end:
```

Finding the largest element



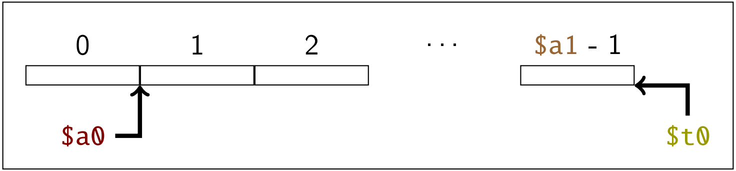

```
        li      $v0, 0              # initial result
        move    $t0, $a1
        slli    $t0, 3              # x « 3 ≡ x × 2^3 = 8x
        add     $t0, $a0            # last addr. plus 64-bit word
loop:
        beq     $t0, $a0, end
        ld      $t1, 0($a0)         # load from memory
⇒       addi    $a0, 8              # advance to next element
        blt     $t1, $v0, loop      # found bigger element?
        move    $v0, $t1
        j       loop
  end:
```

Finding the largest element



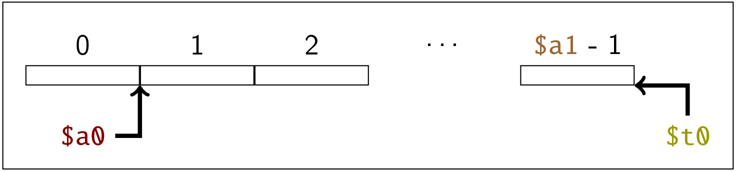

```
        li      $v0, 0              # initial result
        move    $t0, $a1
        slli    $t0, 3              # x « 3 ≡ x × 2^3 = 8x
        add     $t0, $a0            # last addr. plus 64-bit word
loop:
        beq     $t0, $a0, end
        ld      $t1, 0($a0)         # load from memory
        addi    $a0, 8              # advance to next element
⇒       blt     $t1, $v0, loop      # found bigger element?
        move    $v0, $t1
        j       loop
  end:
```

Finding the largest element



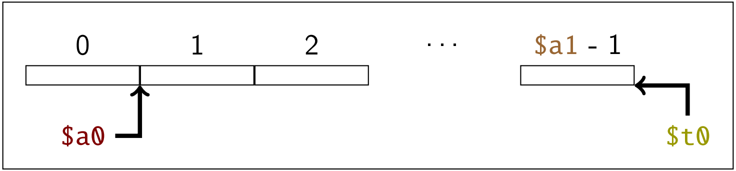

```
         li     $v0, 0            # initial result
         move   $t0, $a1
         slli   $t0, 3            # x « 3 ≡ x × 2^3 = 8x
         add    $t0, $a0          # last addr. plus 64-bit word
loop:
         beq    $t0, $a0, end
         ld     $t1, 0($a0)       # load from memory
         addi   $a0, 8            # advance to next element
         blt    $t1, $v0, loop    # found bigger element?
⇒        move   $v0, $t1
         j      loop
  end:
```

Finding the largest element



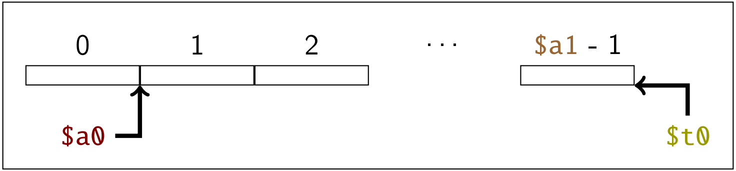

```
        li      $v0, 0              # initial result
        move    $t0, $a1
        slli    $t0, 3              # x « 3 ≡ x × 2^3 = 8x
        add     $t0, $a0            # last addr. plus 64-bit word
loop:
        beq     $t0, $a0, end
        ld      $t1, 0($a0)         # load from memory
        addi    $a0, 8              # advance to next element
        blt     $t1, $v0, loop      # found bigger element?
        move    $v0, $t1
⇒       j       loop
end:
```

Finding the largest element



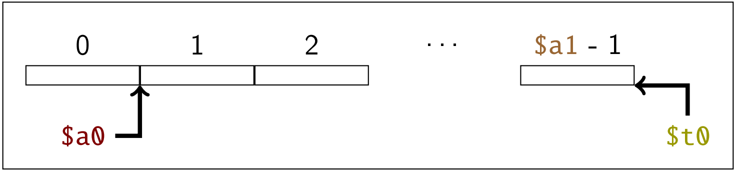

```
        li    $v0, 0            # initial result
        move  $t0, $a1
        slli  $t0, 3            # x « 3 ≡ x × 2³ = 8x
        add   $t0, $a0          # last addr. plus 64-bit word
loop:
⇒       beq   $t0, $a0, end
        ld    $t1, 0($a0)       # load from memory
        addi  $a0, 8            # advance to next element
        blt   $t1, $v0, loop    # found bigger element?
        move  $v0, $t1
        j     loop
end:
```

Finding the largest element



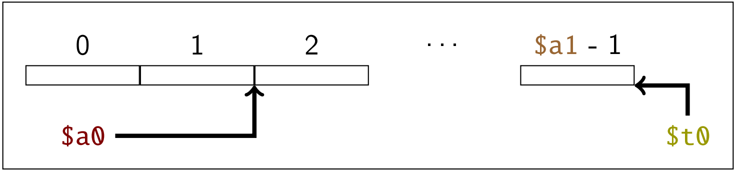

```
        li      $v0, 0              # initial result
        move    $t0, $a1
        slli    $t0, 3              # x « 3 ≡ x × 2^3 = 8x
        add     $t0, $a0            # last addr. plus 64-bit word
loop:
        beq     $t0, $a0, end
⇒       ld      $t1, 0($a0)         # load from memory
        addi    $a0, 8              # advance to next element
        blt     $t1, $v0, loop      # found bigger element?
        move    $v0, $t1
        j       loop
  end:
```

Finding the largest element



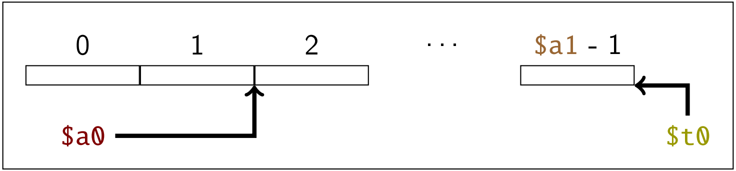

```
        li      $v0, 0            # initial result
        move    $t0, $a1
        slli    $t0, 3            # x « 3 ≡ x × 2^3 = 8x
        add     $t0, $a0          # last addr. plus 64-bit word
loop:
        beq     $t0, $a0, end
        ld      $t1, 0($a0)       # load from memory
⇒       addi    $a0, 8            # advance to next element
        blt     $t1, $v0, loop    # found bigger element?
        move    $v0, $t1
        j       loop
  end:
```

Finding the largest element



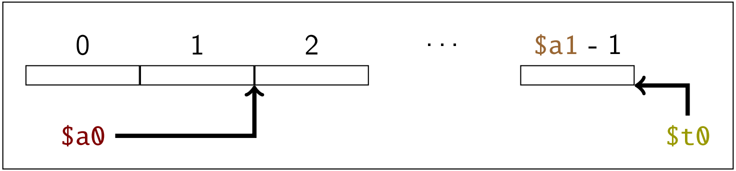

```
          li     $v0, 0          # initial result
          move   $t0, $a1
          slli   $t0, 3          # x « 3 ≡ x × 2^3 = 8x
          add    $t0, $a0        # last addr. plus 64-bit word
loop:
          beq    $t0, $a0, end
          ld     $t1, 0($a0)     # load from memory
          addi   $a0, 8          # advance to next element
⇒         blt    $t1, $v0, loop  # found bigger element?
          move   $v0, $t1
          j      loop
  end:
```

Finding the largest element



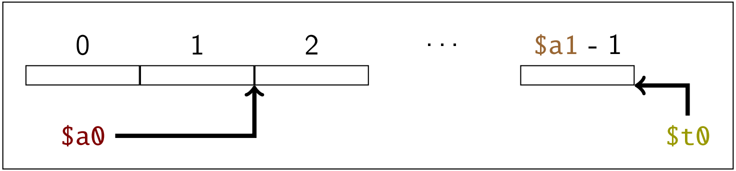

```
        li    $v0, 0            # initial result
        move  $t0, $a1
        slli  $t0, 3            # x « 3 ≡ x × 2^3 = 8x
        add   $t0, $a0          # last addr. plus 64-bit word
loop:
⇒       beq   $t0, $a0, end
        ld    $t1, 0($a0)       # load from memory
        addi  $a0, 8            # advance to next element
        blt   $t1, $v0, loop    # found bigger element?
        move  $v0, $t1
        j     loop
  end:
```

Finding the largest element



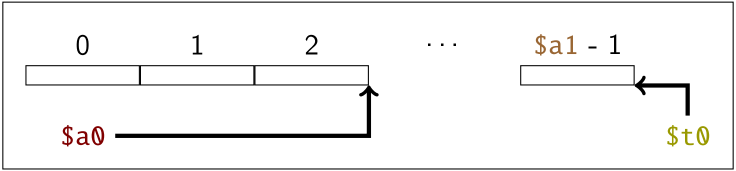

```
          li     $v0, 0             # initial result
          move   $t0, $a1
          slli   $t0, 3             # x « 3 ≡ x × 2^3 = 8x
          add    $t0, $a0           # last addr. plus 64-bit word
loop:
          beq    $t0, $a0, end
⇒         ld     $t1, 0($a0)        # load from memory
          addi   $a0, 8             # advance to next element
          blt    $t1, $v0, loop     # found bigger element?
          move   $v0, $t1
          j      loop
   end:
```

Finding the largest element



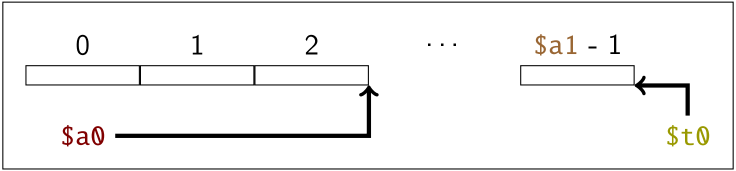

```
        li      $v0, 0              # initial result
        move    $t0, $a1
        slli    $t0, 3              # x « 3 ≡ x × 2^3 = 8x
        add     $t0, $a0            # last addr. plus 64-bit word
loop:
        beq     $t0, $a0, end
        ld      $t1, 0($a0)         # load from memory
⇒       addi    $a0, 8              # advance to next element
        blt     $t1, $v0, loop      # found bigger element?
        move    $v0, $t1
        j       loop
  end:
```

Finding the largest element



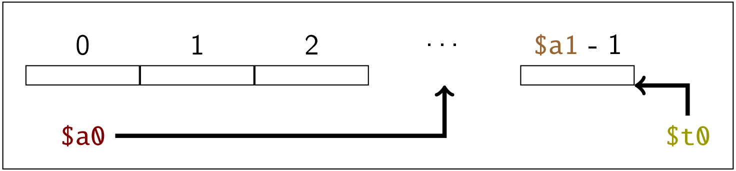

```
        li      $v0, 0              # initial result
        move    $t0, $a1
        slli    $t0, 3              # x « 3 ≡ x × 2^3 = 8x
        add     $t0, $a0            # last addr. plus 64-bit word
loop:
        beq     $t0, $a0, end
        ld      $t1, 0($a0)         # load from memory
        addi    $a0, 8              # advance to next element
⇒       blt     $t1, $v0, loop      # found bigger element?
        move    $v0, $t1
        j       loop
  end:
```

Finding the largest element



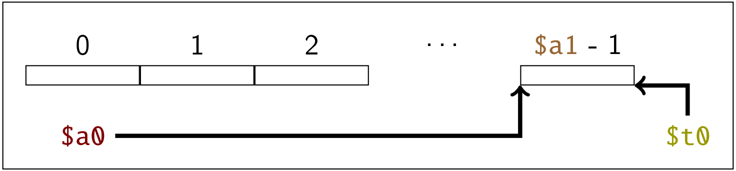

```
        li    $v0, 0              # initial result
        move  $t0, $a1
        slli  $t0, 3              # x « 3 ≡ x × 2^3 = 8x
        add   $t0, $a0            # last addr. plus 64-bit word
loop:
        beq   $t0, $a0, end
        ld    $t1, 0($a0)         # load from memory
        addi  $a0, 8              # advance to next element
        blt   $t1, $v0, loop      # found bigger element?
        move  $v0, $t1
        j     loop
end:
```

Finding the largest element



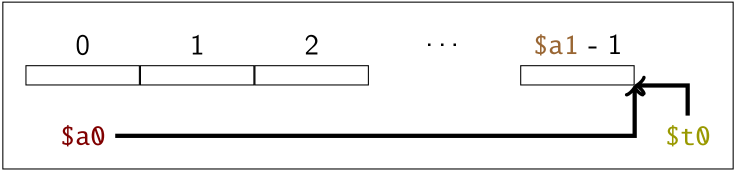

```
        li      $v0, 0              # initial result
        move    $t0, $a1
        slli    $t0, 3              # x « 3 ≡ x × 2^3 = 8x
        add     $t0, $a0            # last addr. plus 64-bit word
loop:
        beq     $t0, $a0, end
        ld      $t1, 0($a0)         # load from memory
        addi    $a0, 8              # advance to next element
        blt     $t1, $v0, loop      # found bigger element?
        move    $v0, $t1
        j       loop
  end:
```

Finding the largest element



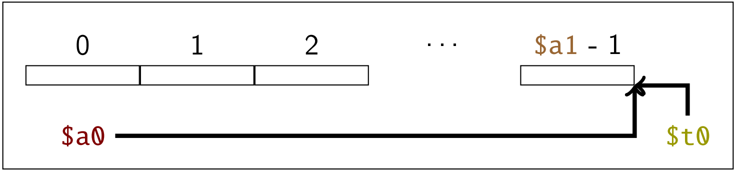

```
        li    $v0, 0              # initial result
        move  $t0, $a1
        slli  $t0, 3              # x « 3 ≡ x × 2^3 = 8x
        add   $t0, $a0            # last addr. plus 64-bit word
loop:
        beq   $t0, $a0, end
        ld    $t1, 0($a0)         # load from memory
        addi  $a0, 8              # advance to next element
        blt   $t1, $v0, loop      # found bigger element?
        move  $v0, $t1
        j     loop
  end:
```

Finding the largest element




```
        li      $v0, 0              # initial result
        move    $t0, $a1
        slli    $t0, 3              # x « 3 ≡ x × 2^3 = 8x
        add     $t0, $a0            # last addr. plus 64-bit word
loop:
⇒       beq     $t0, $a0, end
        ld      $t1, 0($a0)         # load from memory
        addi    $a0, 8              # advance to next element
        blt     $t1, $v0, loop      # found bigger element?
        move    $v0, $t1
        j       loop
end:
```

Finding the largest element




```
        li    $v0, 0            # initial result
        move  $t0, $a1
        slli  $t0, 3            # x « 3 ≡ x × 2^3 = 8x
        add   $t0, $a0          # last addr. plus 64-bit word
loop:
        beq   $t0, $a0, end
        ld    $t1, 0($a0)       # load from memory
        addi  $a0, 8            # advance to next element
        blt   $t1, $v0, loop    # found bigger element?
        move  $v0, $t1
        j     loop
⇒ end:
```

Summary

- Memory access via `ld` (load) and **sd** (store)
- Conditional and unconditional jumps available
  - Conditional jumps for comparing two registers
  - Conditional jumps for comparing one register with zero
- Operational semantics for full assembly language quite complex