CprE 381: Computer Organization and Assembly Level Programming

Henry Duwe

Electrical and Computer Engineering

Iowa State University

Administrative

- Lab3
  - Extra TA office hours due to lab cancellation (for the following dates only in 2050 Coover)
    - **Ashraf: M (2/4) 10am – 11am**
    - **Ryan: F (2/1) 4pm – 6pm**
    - **Trent: W (2/1) 3pm – 4pm**
    - **Rohit: T (2/5) 1pm – 3pm**
    - Normal office hours still in effect
  - Prelab for Lab3 can be turned in with the report
  - Lab 1 and Lab 2 will get leniency on late policy
  - Lab 3 will ***NOT NOT NOT*** have any leniency – it will be graded as late based on Canvas submission

- Lab1
  - Median Student
    - 120 minutes in lab
    - 124 minutes out of lab
  - Report writing was largest portion – probably 1b from comments

Review: While Loops in C

- Consider a **`while`** loop

```
while (A[i] == k)
      i = i + j;
```

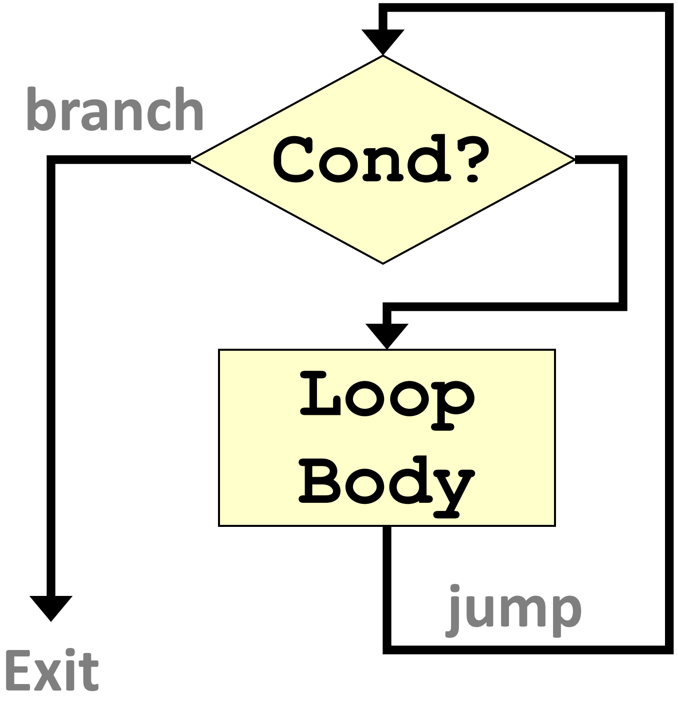
- MIPS assembly loop
- Assume **`i=$s0`**, **`j=$s1`**, **`k=$s2`**, **`&A=$s3`**

```
Loop: sll $t0, $s0, 2       # $t0 = 4 * i
      addu $t1, $t0, $s3    # $t1 = &(A[i])
      lw $t2, 0($t1)        # $t2 = A[i]
      bne $t2, $s2, Exit    # goto Exit if !=
      addu $s0, $s0, $s1    # i = i + j
      j Loop                # goto Loop
   Exit:
```

- Basic block:
  - Maximal sequence of instructions without branches or branch targets

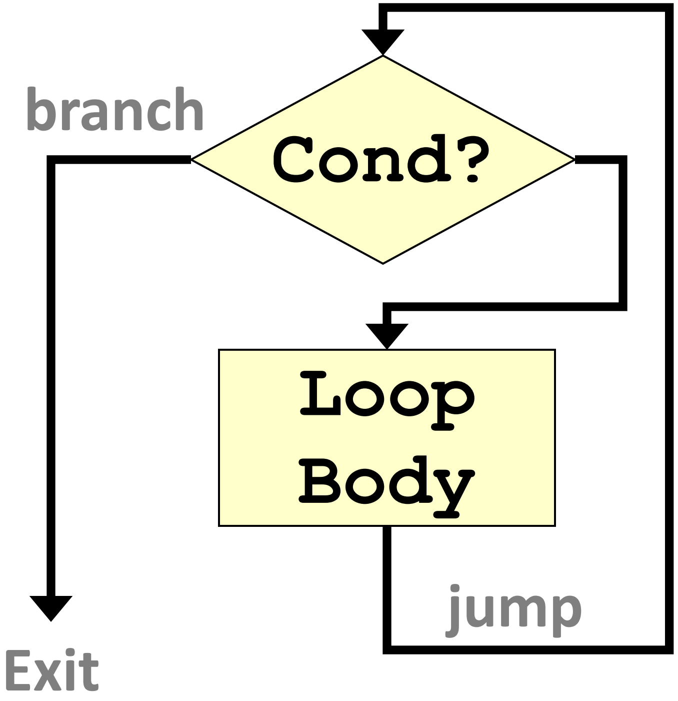
Improve Loop Efficiency

- Code uses two branches per iteration:

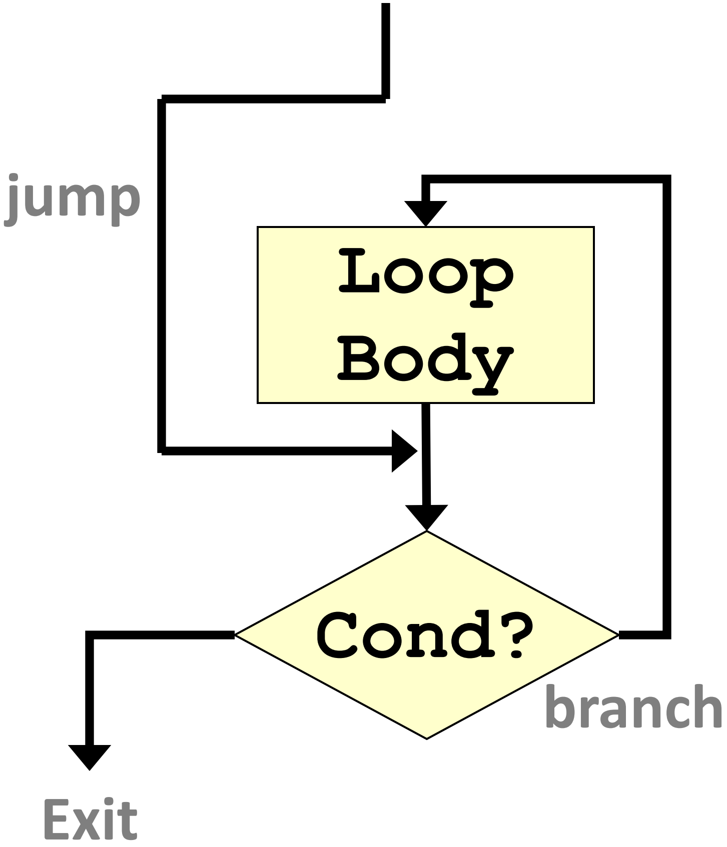


Improve Loop Efficiency

- Code uses two branches per iteration:



- More efficient structure:



Improved Loop Solution

- Remove extra jump from loop body

```
    j Cond                      # goto Cond
Loop: addu $s0, $s0, $s1       # i = i + j
Cond: sll $t0, $s0, 2          # $t0 = 4 * i
    addu $t1, $t0, $s3         # $t1 = &(A[i])
    lw $t2, 0($t1)             # $t2 = A[i]
    beq $t2, $s2, Loop         # goto Loop if ==
Exit:
```

- Reduced loop from 6 to 5 instructions
  - Even small improvements important if loop executes frequently and for many iterations

For Loops in C

- Consider a `for` loop

```
for (i=0; i<32; i++)
    a[i] = b[i] + j;
```

- MIPS assembly loop
  - Assume: `char a[32]; char b[32]; char j;`
  - Assume `i=$s0`, `j=$s1`, `b=$s2`, `a=$s3`

For Loops in C

- Consider a **for** loop

```
for (i=0; i<32; i++)
      a[i] = b[i] + j;
```

- MIPS assembly loop
  - Assume: **char a[32]; char b[32]; char j;**
  - Assume **i=$s0**, **j=$s1**, **b=$s2**, **a=$s3**

```
       sub $s0, $s0, $s0       # i=0
       j Cond                  # goto Cond
Loop:  addu  $t2, $s2, $s0     # $t2 = &(b[i])
       lb    $t1, 0($t2)       # b[i]
       addu  $t0, $t1, $s1     # $t0 = b[i] + j
       addu  $t3, $s3, $s0     # $t3 = &(a[i])
       sb    $t0, 0($t3)       # a[i] = $t0
       addui $s0, $s0, 1       # i++
Cond:  slti  $t0, $s0, 32      # (i<32)
       bne   $t0, $0, Loop     # goto Loop if i<32
```

For Loops in C – Less Control Overhead

- New `for` loop

```
for (i=0; i<32; i+=4) {
     a[i] = b[i] + j;
     a[i+1] = b[i+1] + j;
     a[i+2] = b[i+2] + j;
     a[i+3] = b[i+3] + j;
}
```

- Called **loop unrolling**

For Loops in C – Less Control Overhead

- MIPS assembly loop

```
      sub  $s0, $s0, $s0   # i=0
      j Cond               # goto Cond
Loop: addu $t2, $s2, $s0   # $t2 = &(b[i])
      addu $t3, $s3, $s0   # $t3 = &(a[i])
      lb   $t1, 0($t2)     # b[i]
      addu $t0, $t1, $s1   # $t0 = b[i] + j
      sb   $t0, 0($t3)     # a[i] = $t0
      lb   $t1, 1($t2)     # b[i+1]
      addu $t0, $t1, $s1   # $t0 = b[i+1] + j
      sb   $t0, 1($t3)     # a[i+1] = $t0
      lb   $t1, 2($t2)     # b[i+2]
      addu $t0, $t1, $s1   # $t0 = b[i+2] + j
      sb   $t0, 2($t3)     # a[i+2] = $t0
      lb   $t1, 3($t2)     # b[i+3]
      addu $t0, $t1, $s1   # $t0 = b[i+3] + j
      sb   $t0, 3($t3)     # a[i+3] = $t0
      addu $s0, $s0, 4     # i+=4
Cond: slti $t0, $s0, 32    # (i<8)
      bne $t0, $0, Loop    # goto Loop if i<32
```

For Loops in C – Less Control Overhead

- MIPS assembly loop

```
      sub  $s0, $s0, $s0   # i=0
      j Cond               # goto Cond
Loop: addu $t2, $s2, $s0   # $t2 = &(b[i])
      addu $t3, $s3, $s0   # $t3 = &(a[i])
```

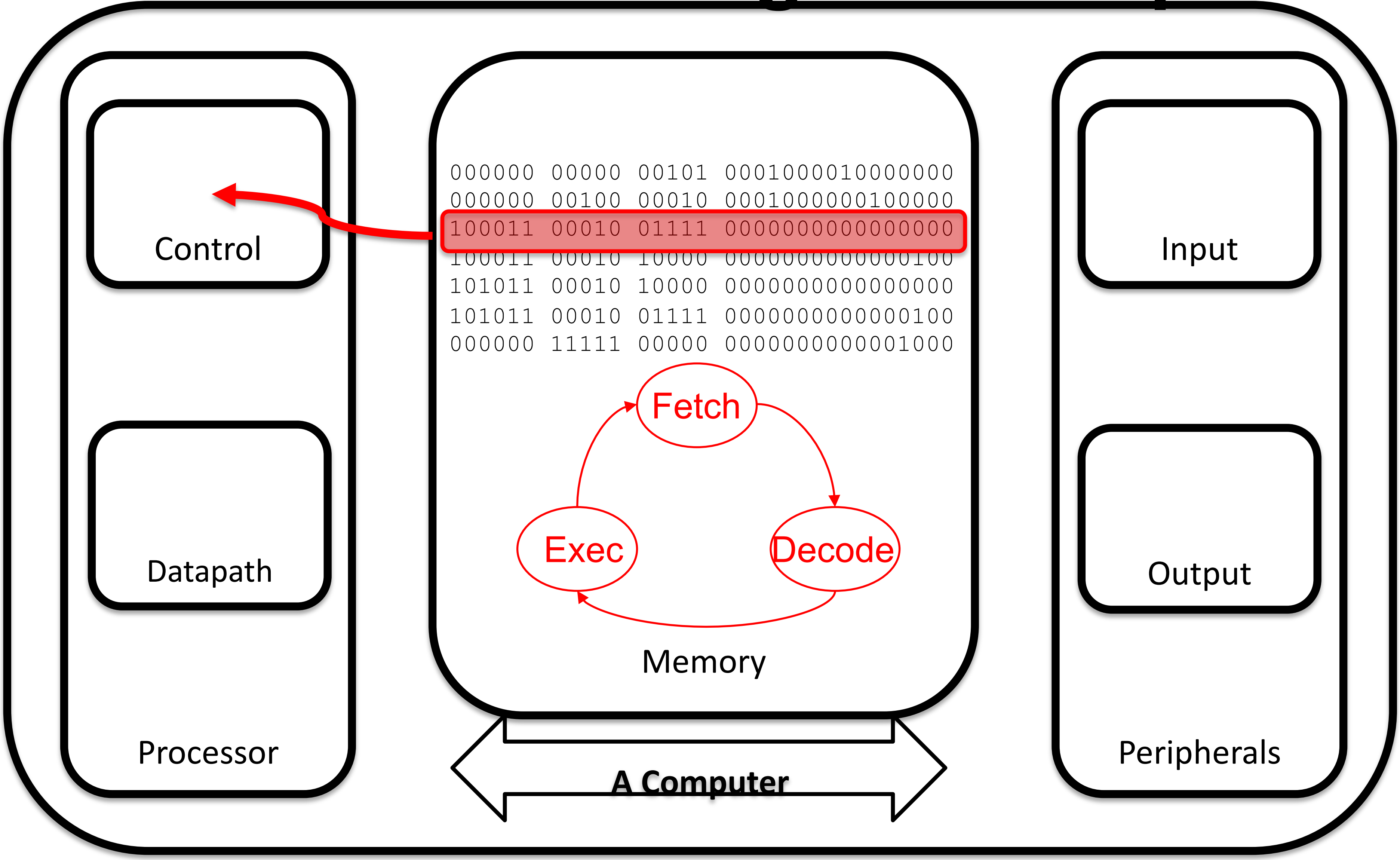
In-class Assessment!

Access Code: Flynn

Note: sharing access code to those outside of classroom or using access while outside of classroom is considered cheating

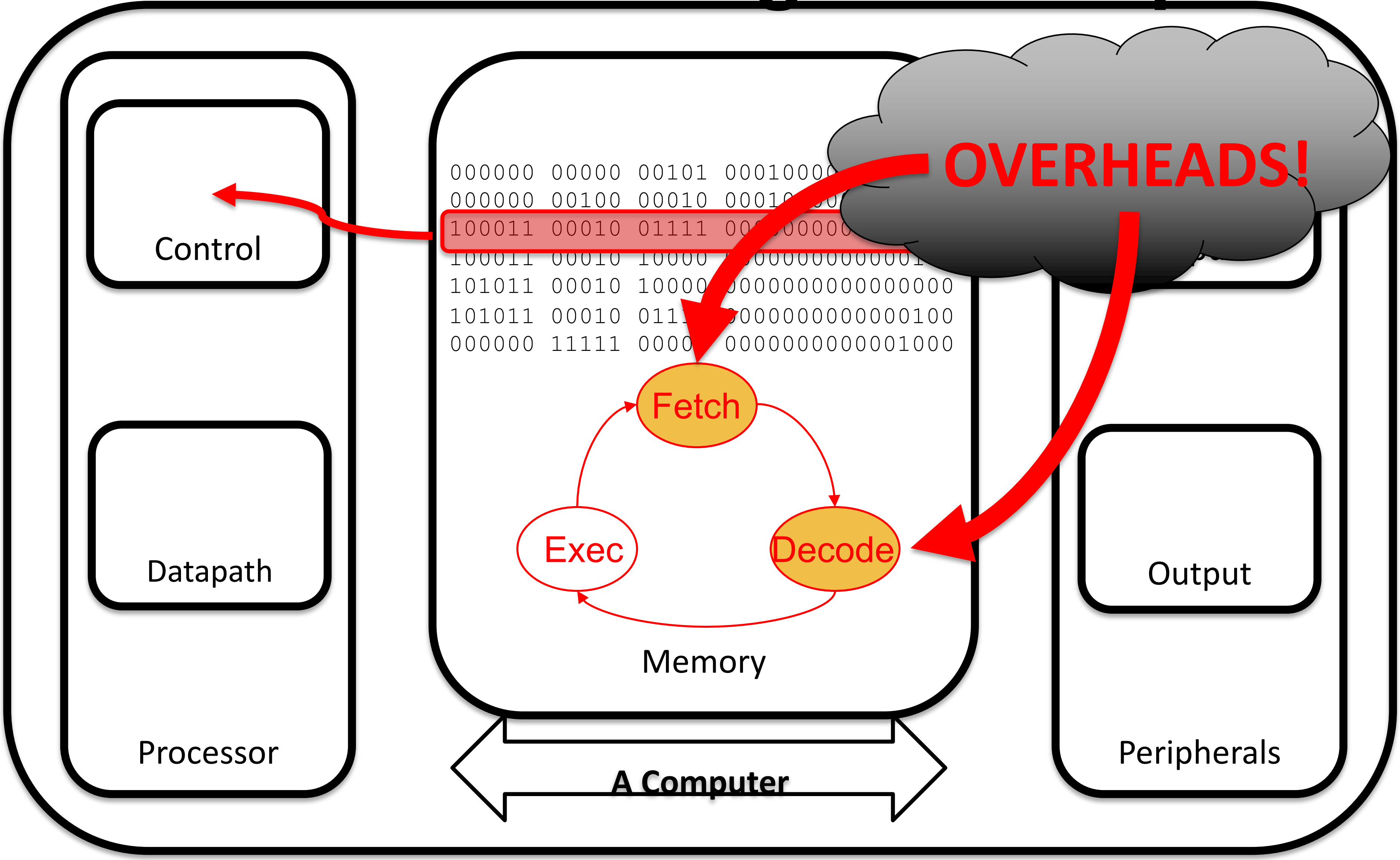
```
      sb   $t0, 2($t3)     # a[i+2] = $t0
      lb   $t1, 3($t2)     # b[i+3]
      addu $t0, $t1, $s1   # $t0 = b[i+3] + j
      sb   $t0, 3($t3)     # a[i+3] = $t0
      addu $s0, $s0, 4     # i+=4
Cond: slti $t0, $s0, 32    # (i<8)
      bne $t0, $0, Loop    # goto Loop if i<32
```

Review: Stored Program Computer



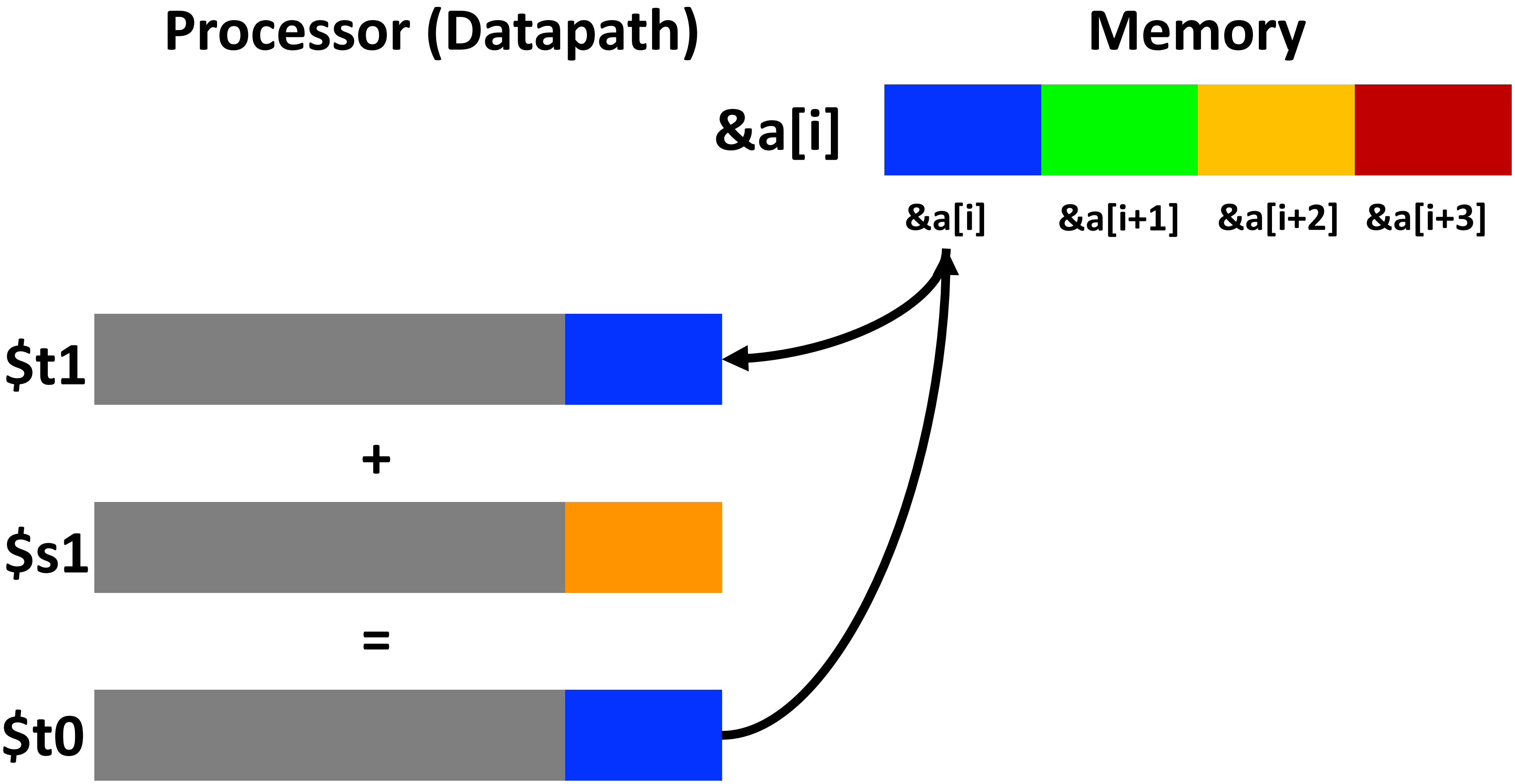
```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
100011 00010 01111 0000000000000000
100011 00010 10000 0000000000000100
101011 00010 10000 0000000000000000
101011 00010 01111 0000000000000100
000000 11111 00000 0000000000001000
```

Review: Stored Program Computer

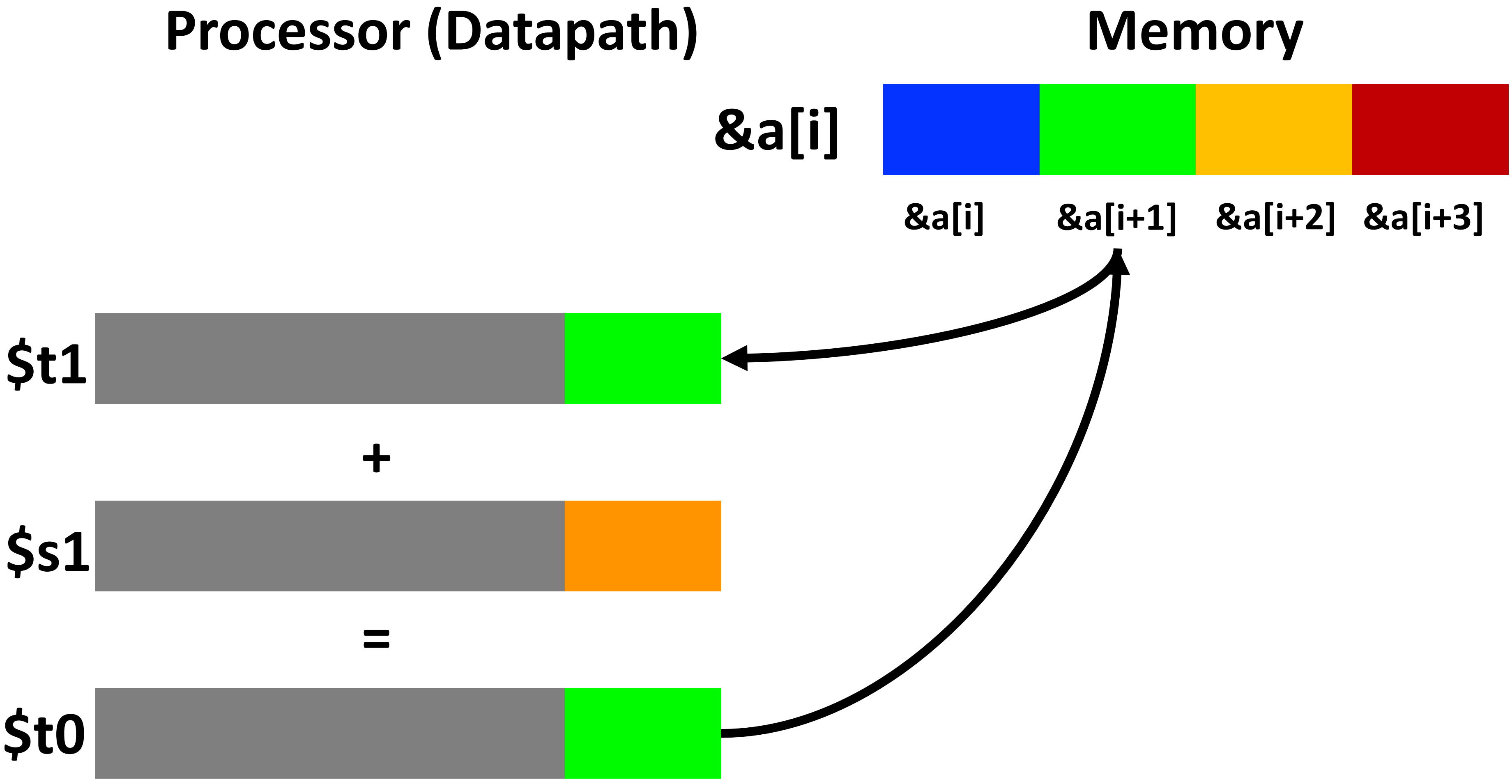


```
000000 00000 00101 00010000
000000 00100 00010 00010
100011 00010 01111 00000000
100011 00010 10000 0000000000000001
101011 00010 10000 0000000000000000
101011 00010 0111 0000000000000100
000000 11111 0000  0000000000001000
```

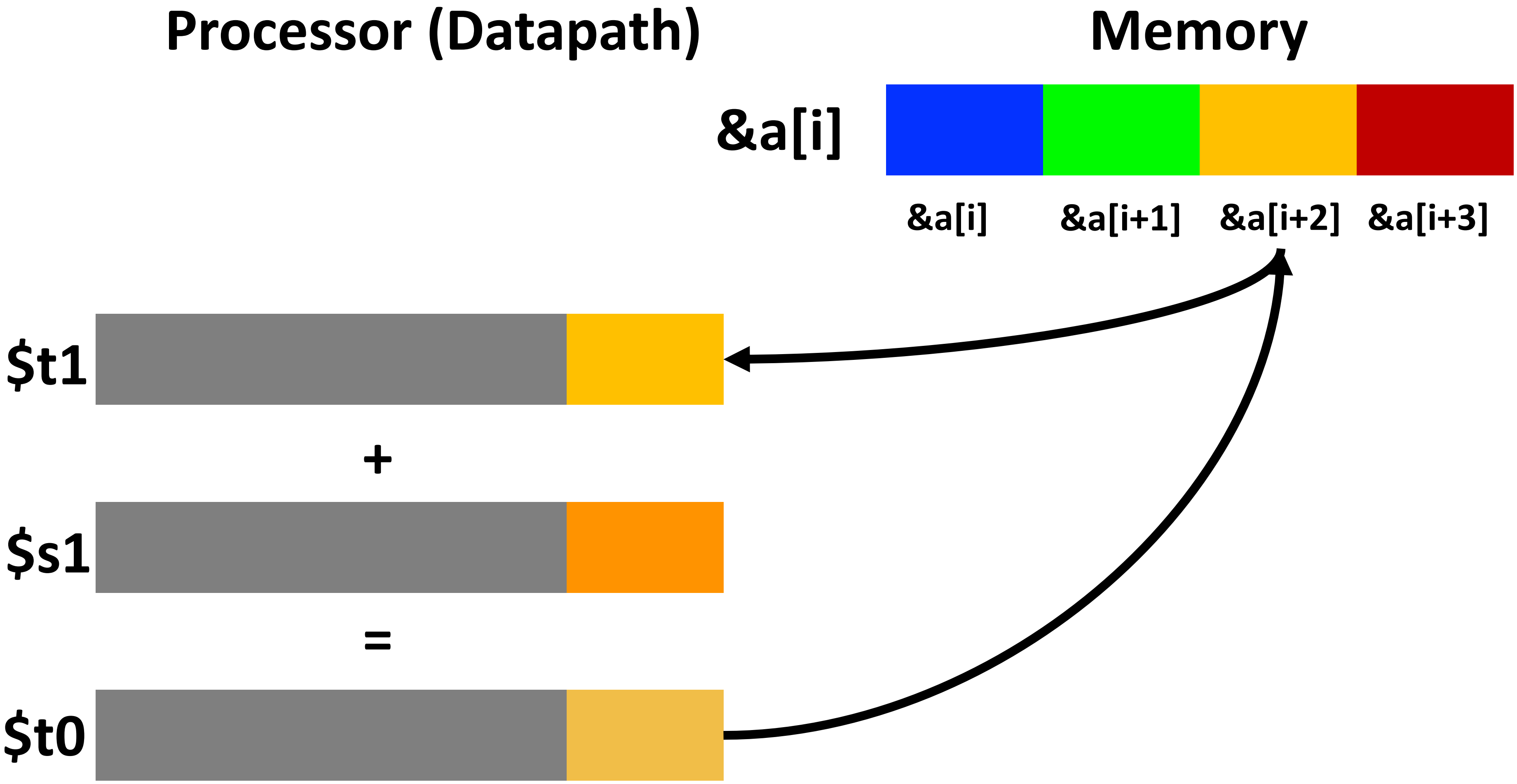
Visualization



Visualization



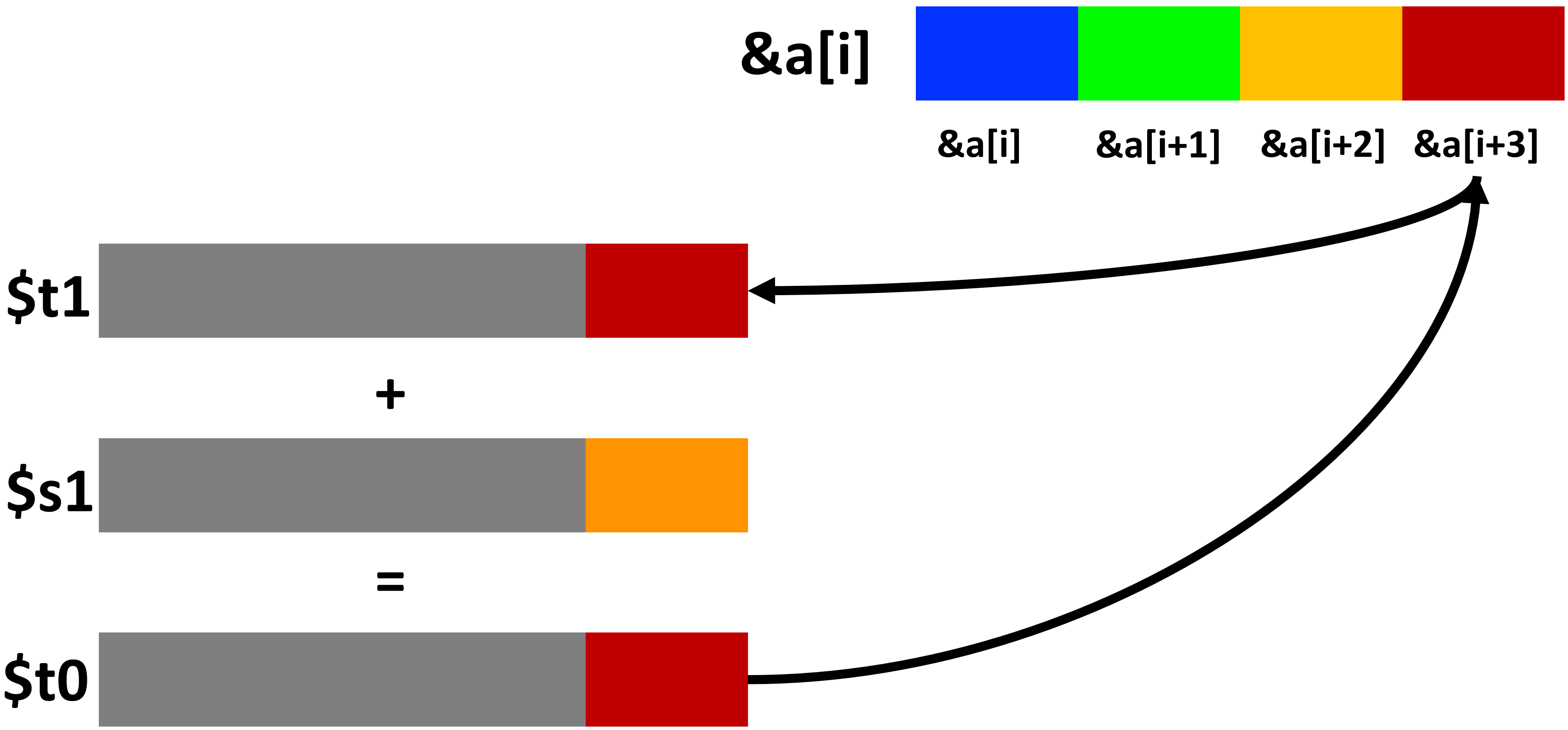
Visualization



Visualization

Processor (Datapath)

Memory



&a[i]

&a[i] &a[i+1] &a[i+2] &a[i+3]

$t1

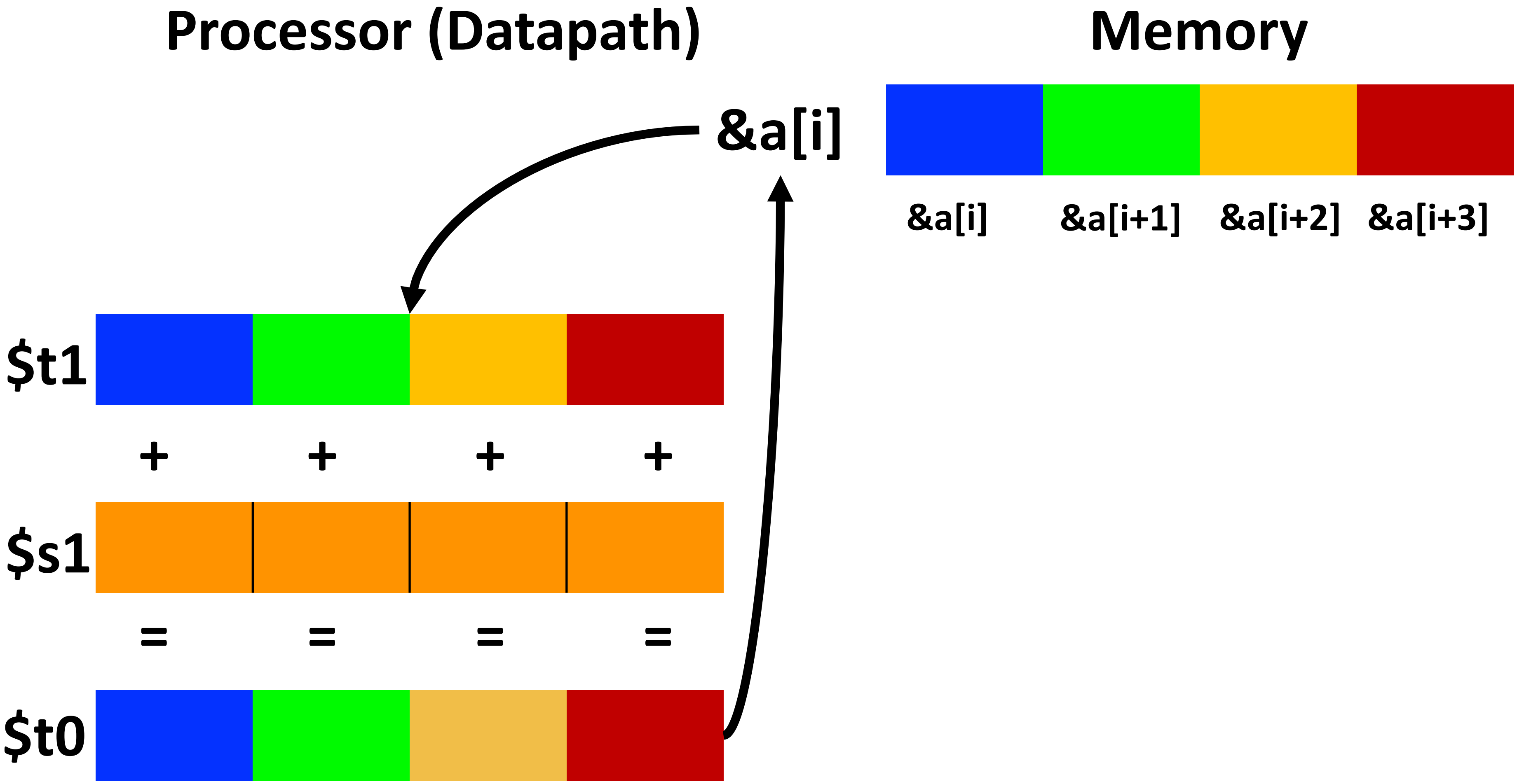
+

$s1

=

$t0

Visualization – SIMD



Even Less Control Overhead

- MIPS assembly loop
  - Using MIPS digital signal processing (DSP) application-specific extension (ASE)
  - Form of subword **SIMD** (single instruction, multiple data)
  - || means concatenation in the comments
  - .qb means treat the registers as quad bytes (i.e., four independent byte values concatenated)

```
      replv.qb $s1, $s1      # $s1 now stores j||j||j||j
      sub  $s0, $s0, $s0     # i=0
      j Cond                 # goto Cond
Loop: addu $t2, $s2, $s0     # $t2 = &(b[i])
      addu $t3, $s3, $s0     # $t3 = &(a[i])
      lw   $t1, 0($t2)       # b[i]||b[i+1]||b[i+2]||b[i+3]
      addu.qb $t0, $t1, $s1  # $t0 = (b[i]+j)||(b[i+1]+j)||...
      sw   $t0, 0($t3)     # a[i]=$t0[31:24],a[i+1]=$0[23:16]...
      addu $s0, $s0, 1       # i+=4
Cond: slti $t0, $s0, 8       # (i<8)
      bne $t0, $0, Loop      # goto Loop if i<32
```

For Loops in C -- Summary

- Consider a **`for`** loop

```
for (i=0; i<32; i++)
    a[i] = b[i] + j;
```

- MIPS assembly loops -- functionally equivalent

Version	# Static Instructions	# Dynamic Instructions
Base	10	260
Unrolled 4x	19	140
SIMD 4x	11	69

HW Preview: What about `switch`?

- Consider:

```
  switch(x) {
      case 2:
          y = x << 1;
          x++;
          break;
      case 1:
          y = x;
          x++;
          break;
      case 0:
          y = 0;
      default:
          x = 0;
 }
```

Acknowledgments

- These slides contain material developed and copyright by:
  - – Joe Zambreno (Iowa State)
  - – David Patterson (UC Berkeley)
  - – Mary Jane Irwin (Penn State)
  - – Christos Kozyrakis (Stanford)
  - – Onur Mutlu (Carnegie Mellon)
  - – Krste Asanović (UC Berkeley)