TU WIEN Informatics

Advanced Computer Architecture

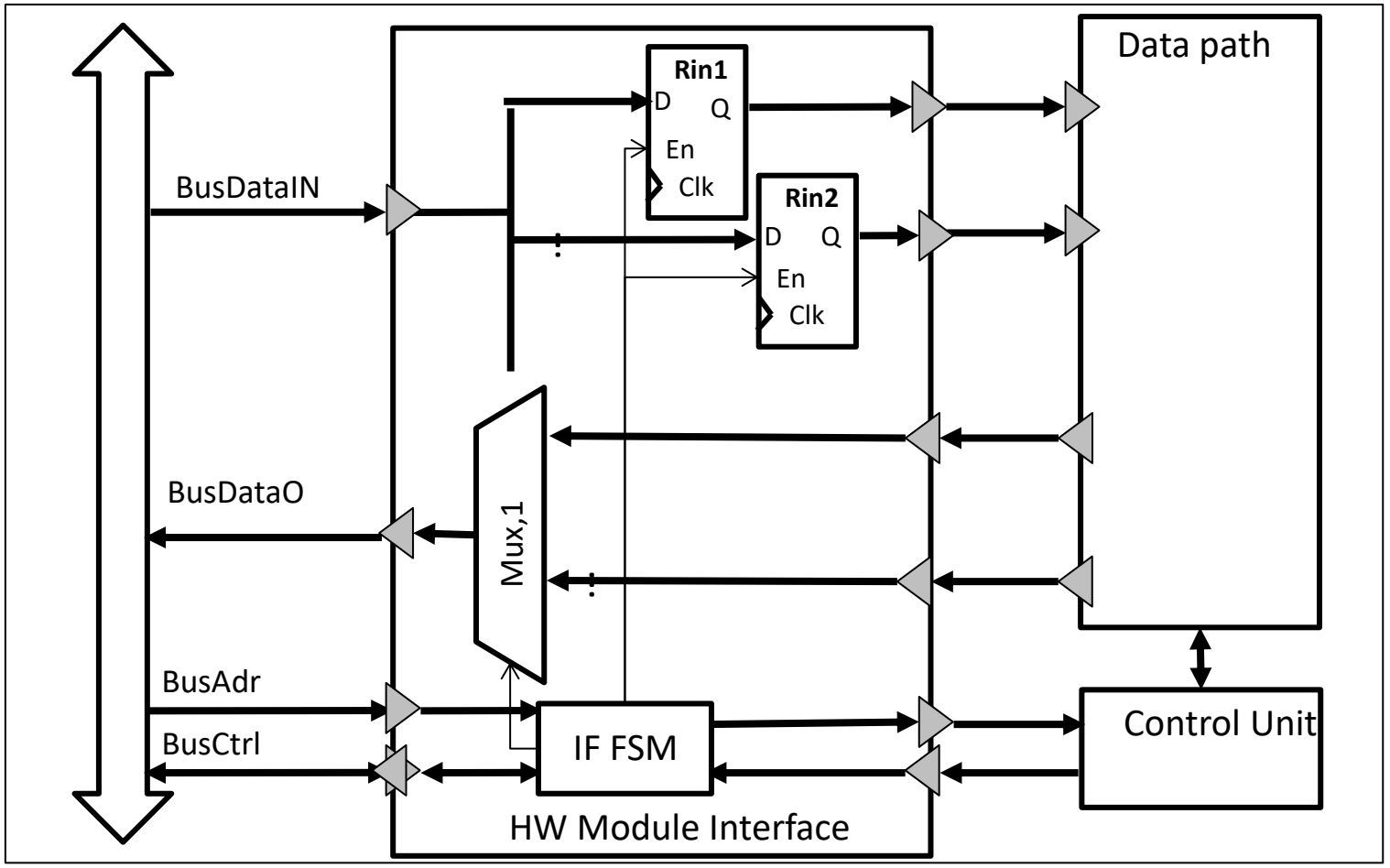
D4 –High Level Synthesis (HLS) – I/O and Loop Optimizations

Daniel Mueller-Gritschneder

D4-1 I/O Scheduling

Register Interface

- Data ist stored in Registers in the Interface (e.g. adressable via bus)
- Read/Write operations can be scheduled concurrently with zero delay

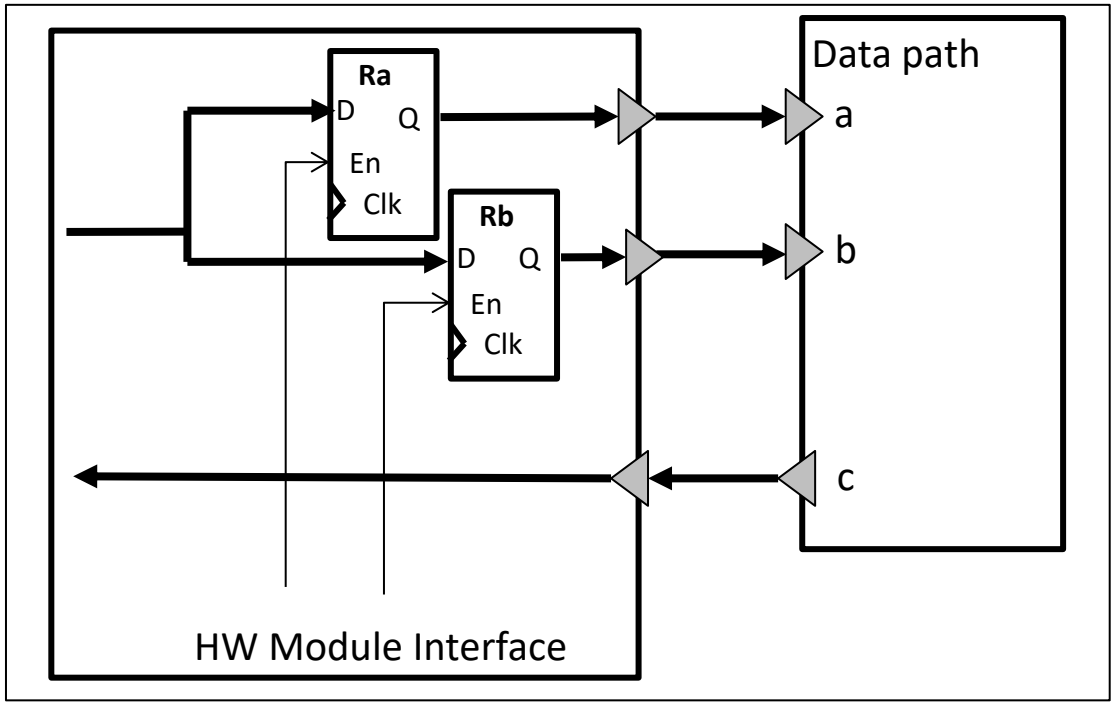


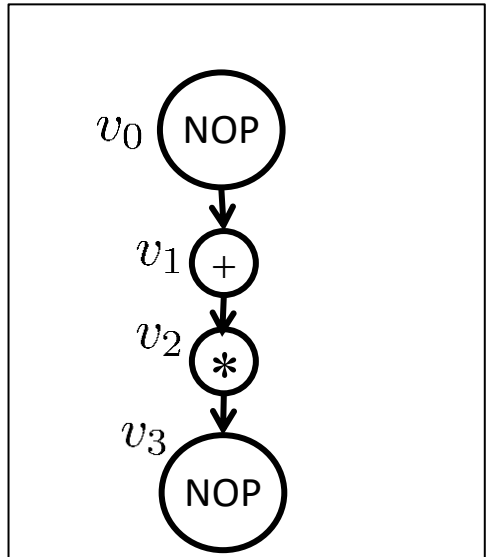
Register Interface

- Example: Read/Write operations scheduled implicitly.

C-Code section

```
int binomial(int a, int b) {
        int c=a+b;
         c=c*c;
        return c;
}
```



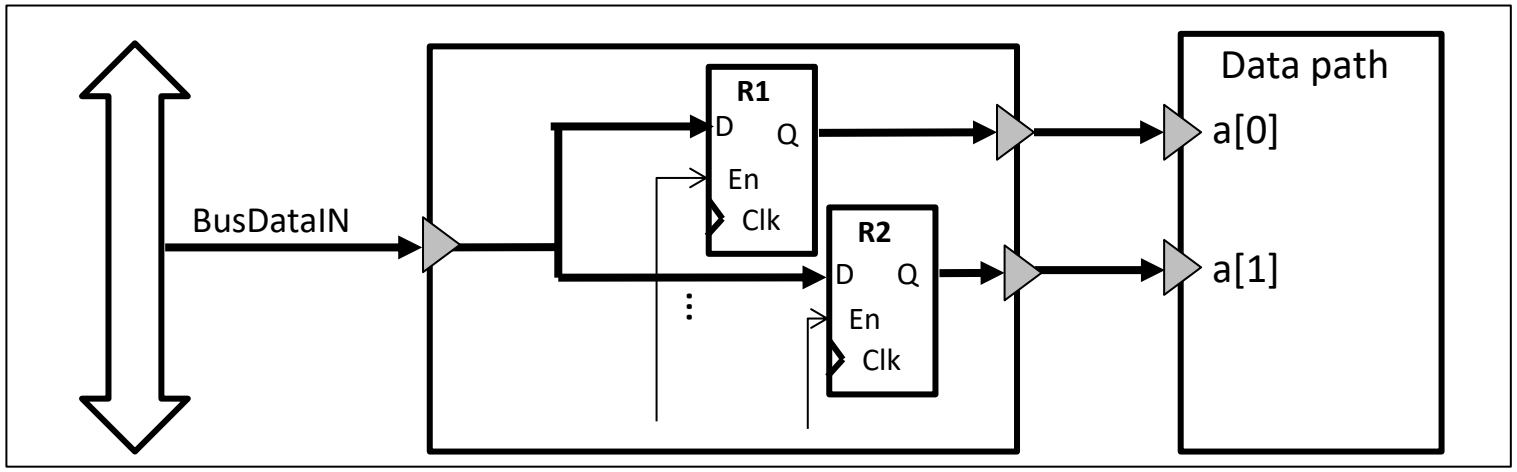


Array Register Interface

- Array is stored in registers in the interface (e.g. adressable via bus)
- Read/Write operations can be scheduled concurrently with zero delay
- Example:

C-Code section

```
int acc(int a[4]) {
        int c1=  a[0]+a[1];
         int c2=  a[2]+a[3];
        int c=c1+c2;
        return c;
}
```

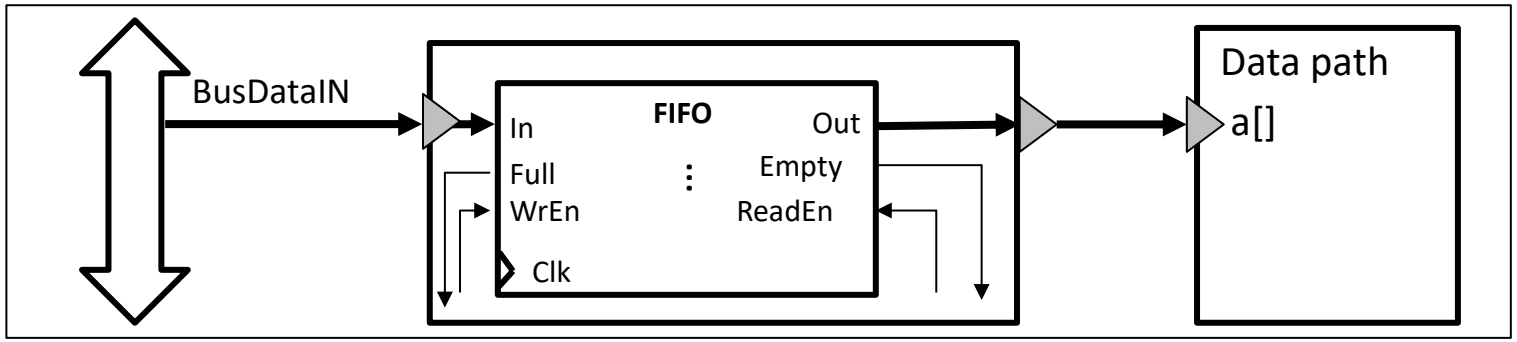


Array FIFO Interface

- Array is stored in FIFO buffer in the interface (e.g. adressable via bus)
- Read/Write operations can be scheduled only sequencially with zero delay
- Stalls possible if FIFO is empty
- Example:

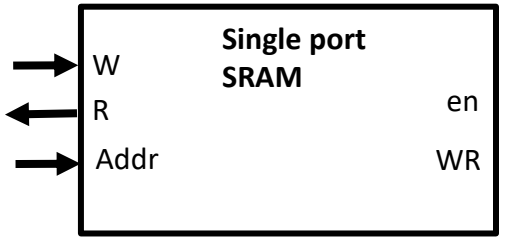
C-Code section

```
int acc(int a[4]) {
        int c1=  a[0]+a[1];
         int c2=  a[2]+a[3];
        int c=c1+c2;
        return c;
}
```



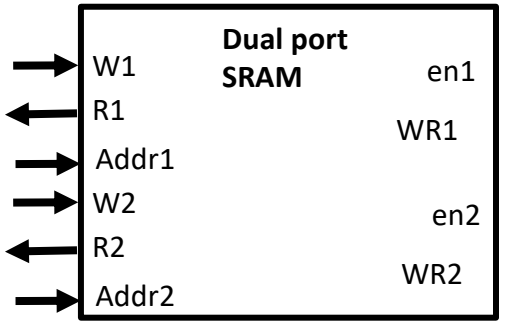
SRAM buffers

- On chip buffers using SRAM cells (different from flip-flops)
- Single-port SRAM
  - Only one port to read or write



- Dual-port SRAM
  - Two ports to read or write
  - Cannot read/write same location on both ports at same time
  - True dual port SRAM: Can read same location on both ports, writes or read/write still needs to be abitrated

- Timing
  - Return data either in same (zero delay) or next clock cycle (pipelined)

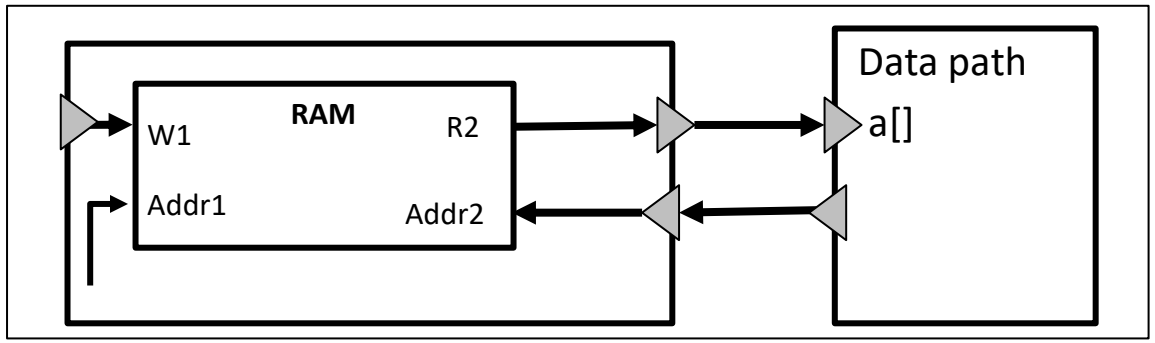


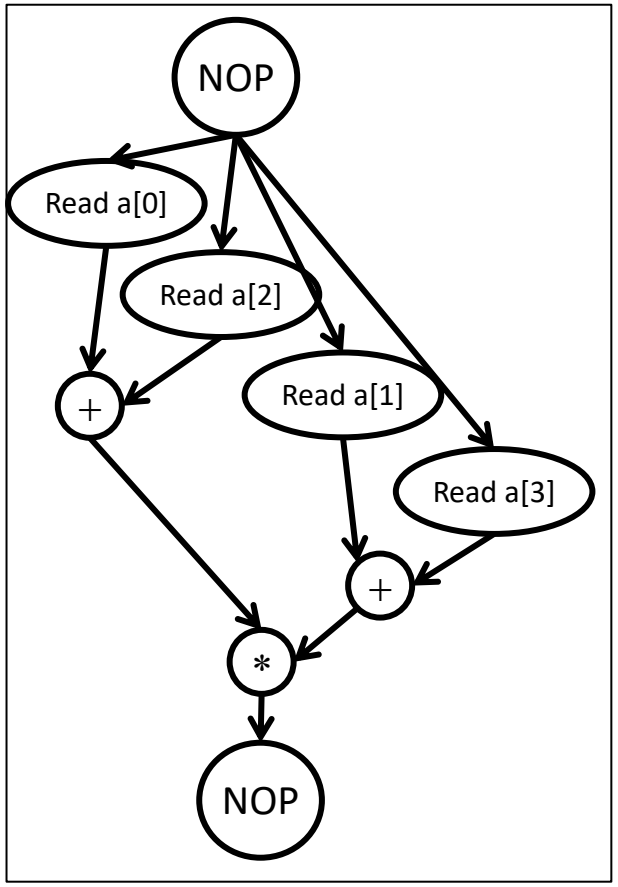
Input Array Memory Interface

- Array is stored in RAM block in the interface (e.g. adressable via bus)
- Read/Write operations can be scheduled only one per cycle to diff. address

C-Code section

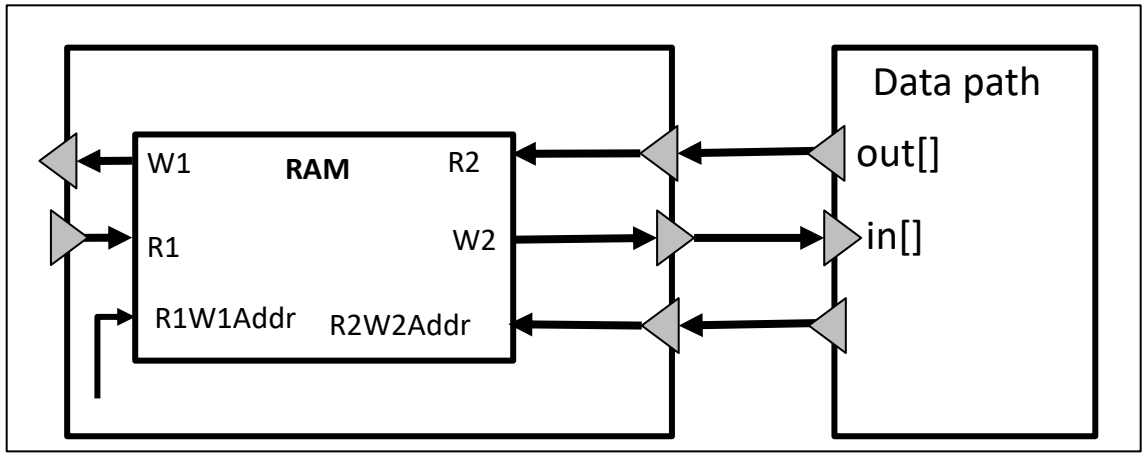
```
int acc(int a[4]) {
        int c1=  a[0]+a[2];
         int c2=  a[1]+a[3];
        int c=c1*c2;
        return c;
}
```





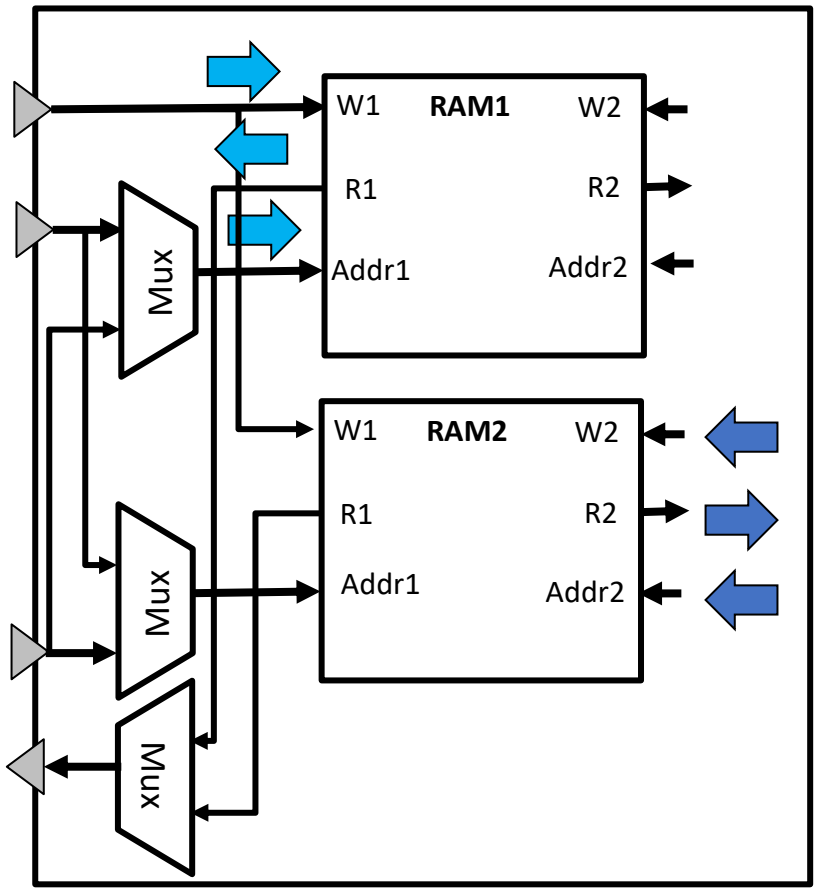
Input/Output Array Memory Interface

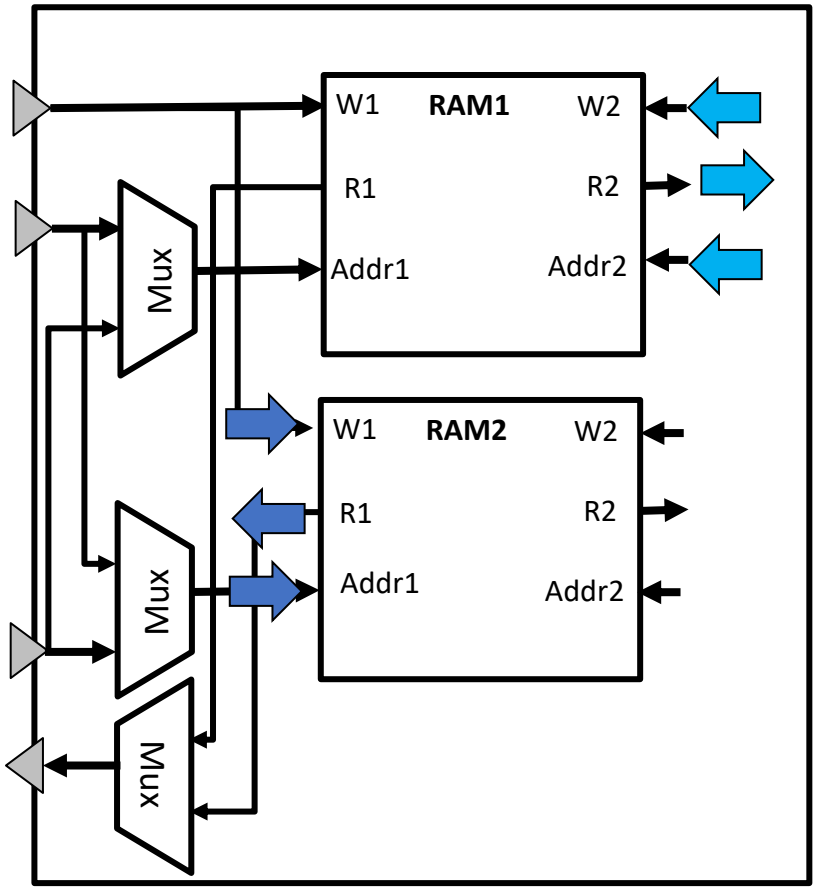
- Array is stored in RAM block in the interface (e.g. adressable via bus)
- Dual-port: Read/Write operations (R1+W1/R2+W2) can be scheduled only one per cycle and port 1 and port 2 need different addresses
- True dual-port: Read operations (R1/R2) can read same address in parallel



Ping Pong Array Memory Interface

- One RAM for input one RAM for output
- Switch/overlap between phases (ping pong scheme)
- All ports can be kept busy if read/writes from two different execution runs overlap (high utilization of memory ports).





D4-2 Control Flow and Loop Scheduling

Combining Schedules of SGUs

- Algorithms find schedule for a single sequencing graph unit (SGU).
- Hierarchy nodes (CALL, BR, LOOP) represent a SGU.
- Schedule for complete sequencing graph is found by:
  - Compute schedule for SGU on lowest level of hierarchy first.
  - Extract execution time of hierarchy nodes from latency of schedule of their corresponding SGUs.
  - Schedule top level SGU with hierarchy node.
  - Shift start time of schedule of lower level SGU to start time of corresponding hierarchy node.
- Schedule can be data dependent or independent.

Execution time of hierarchy nodes

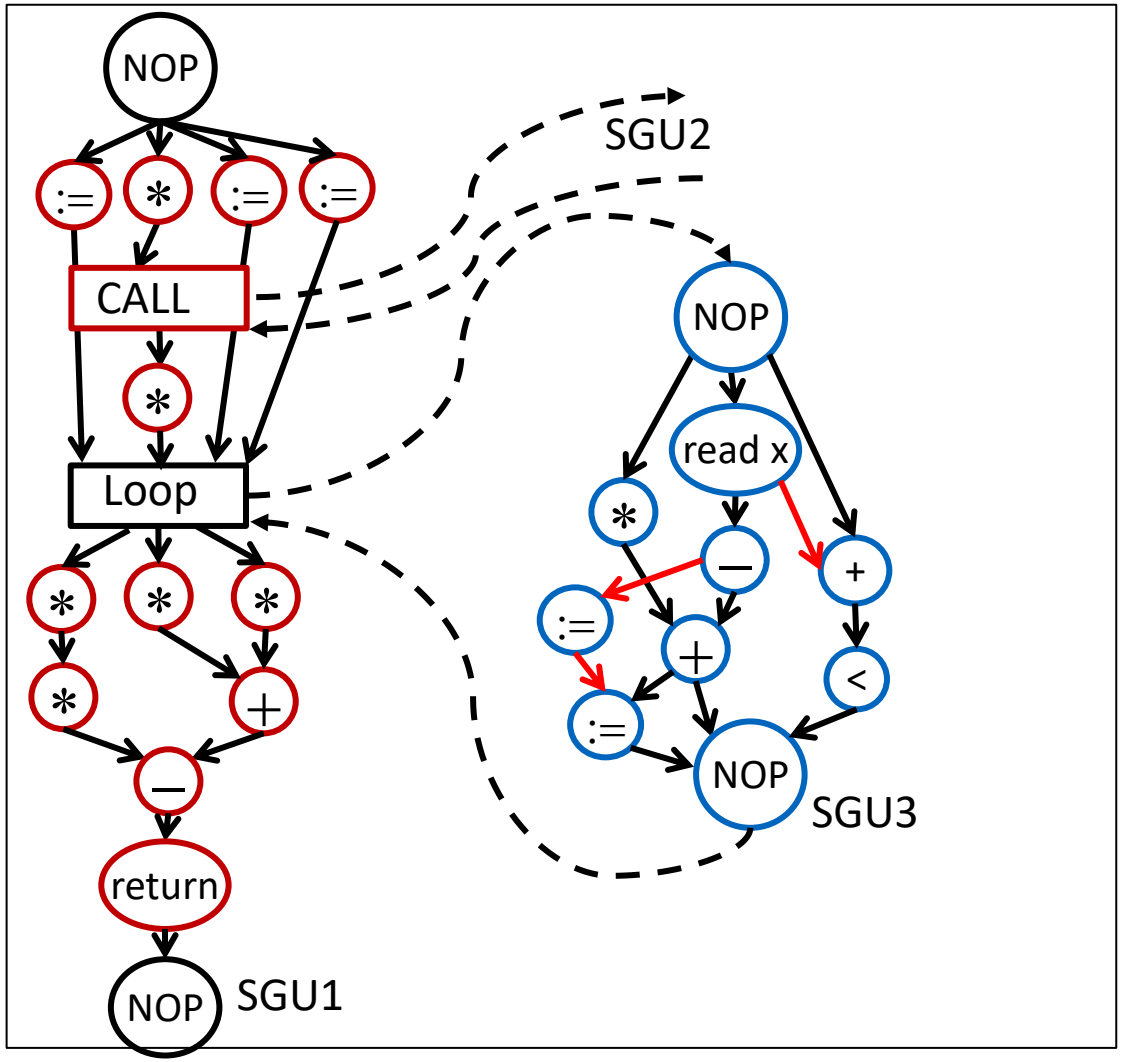
- Usually only support for data-independent schedule

- CALL node: Execution time equals to latency of schedule of corresponding SGU in the lower hierarchy.
- BR node (branch): Execution time equals to maximal latency of all corresponding SGUs in the lower hierarchy.
- LOOP node:
  - Requirement: Fixed number of loop iterations
  - Execution delay equals latency of SGU of lower hierarchy times number of loop iterations.

Execution time of hierarchy nodes

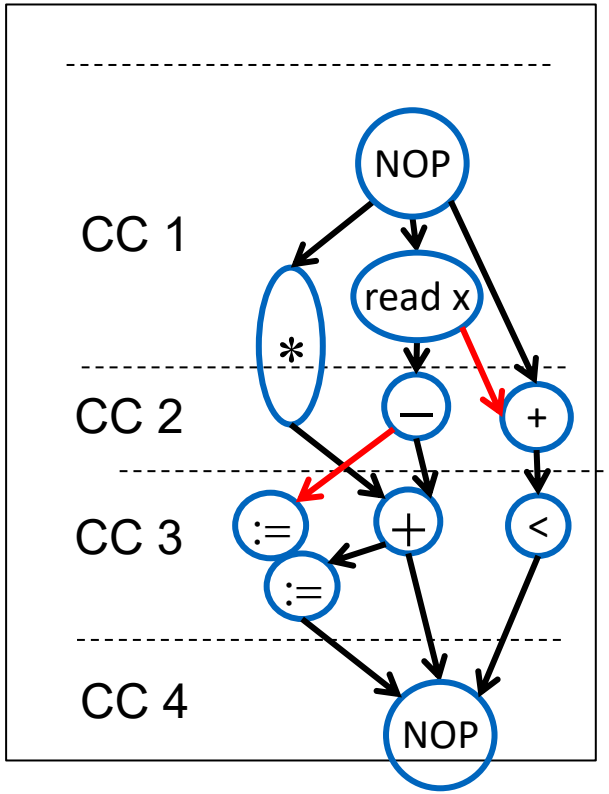
- Example: Goertzel algorithm

```
B1: s_prev1 := 0.0
    s_prev2 := 0.0
    i:=0
    t1 := 6.28
    f := t1 * freq
    param f
    t2 := call cos,1
    coeff:=2.0*t2

B2: t3:= coeff * s_prev1
    t4:= x[i]
    t5 := t4 - s_prev2
    s := t3 + t5
    s_prev2 := s_prev1
    s_prev1 := s
    i:=i+1
    if i < 64 goto B2

B3: t6:= s_prev1 * s_prev1
    t7:= s_prev2 * s_prev2
    t8:= s_prev1 * s_prev2
    t9:= t8 * coeff
    t10:= t6+t7
    power:= t10 - t9
    return power
```



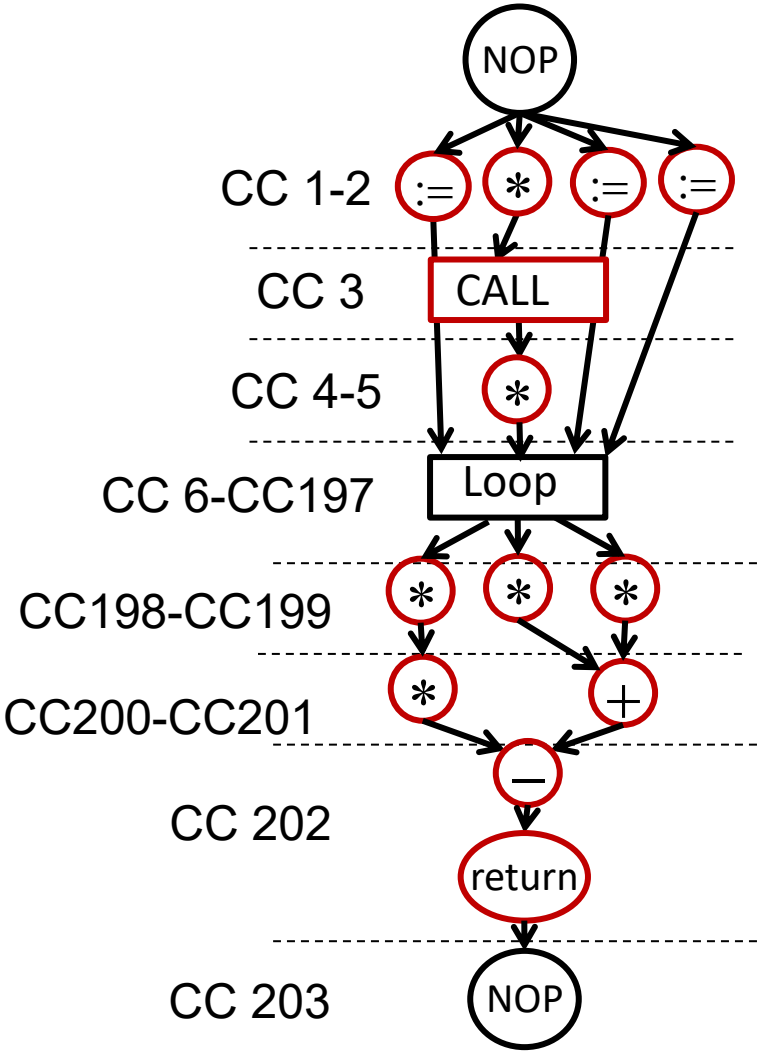
Execution time of hierarchy nodes

- Example: Goertzel-Algorithm
  - ASAP Schedule
  - Operation delays:
    - 0 clock cycles for :=
      (Both := merged to allow scheduling in same cycles)
    - 1 clock cycles for -,+,incr,>,read
    - 2 clock cycles for *
  - Latency $\Lambda_{SGU3}=3$



Execution time of hierarchy nodes

- Example: Goertzel algorithm
  - ASAP Algorithm
  - Execution delays:
    - 0 cycles for :=
    - 1 cycle for -,+,incr,>
    - 2 cycle for *
    - CALL node:
      - 1 cycle for cos
      - Lookup table
    - LOOP node:
      - 64 loop iterations
      - $\Lambda_{SGU3}=3$
      - 3*64=192 cycles



Scheduling Do-all Loops

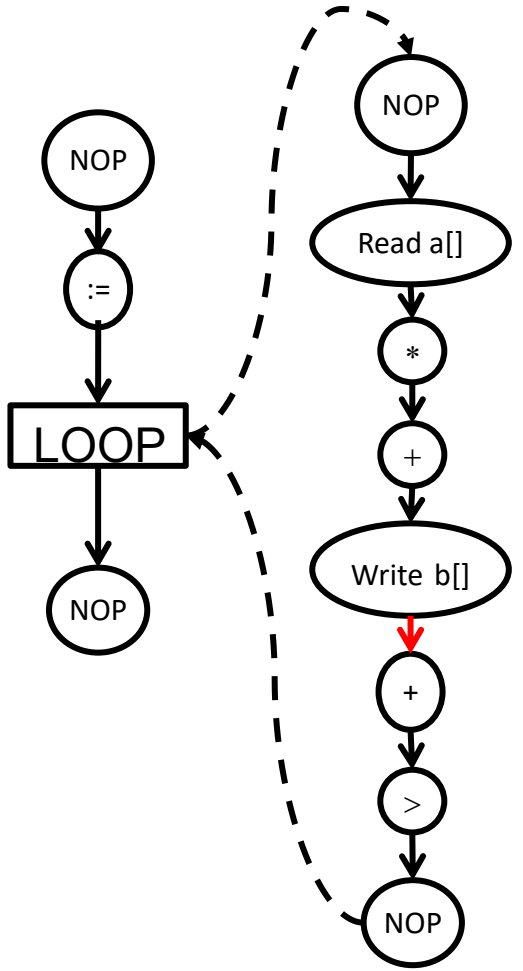
- Example:

C-Code section

```
void loopex(int* a,int* b)
{
     for (int i=0;i<20;i++)

          b[i]=a[i]*a[i]+a[i];
}
```

Three-address code

```
loopex:
i:=0
B1:t1:=a[i]
   t2:=t1*t1
   t3:=t1+t2
   b[i]:=t3
   i=i+1
   if (i<20) goto B1
```



Scheduling Do-all Loops

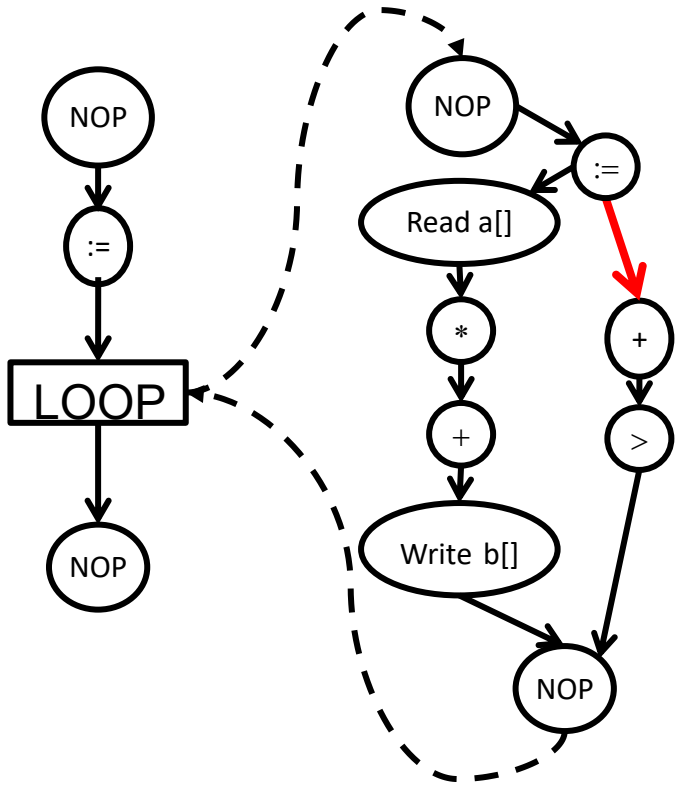
- Apply tree-height reduction (THR)

Three-address code

```
loopex:
i:=0
B1:t1:=a[i]
   t2:=t1*t1
   t3:=t1+t2
   b[i]:=t3
   i=i+1
   if (i<20) goto B1
```

Three-address code with THR

```
loopex:
i:=0
B1:t1:=i  //THR
   t2:=a[t1]
   t3:=t2*t2
   t4:=t2+t3
   b[t1]:=t4
   i=i+1
   if (i<20) goto B1
```

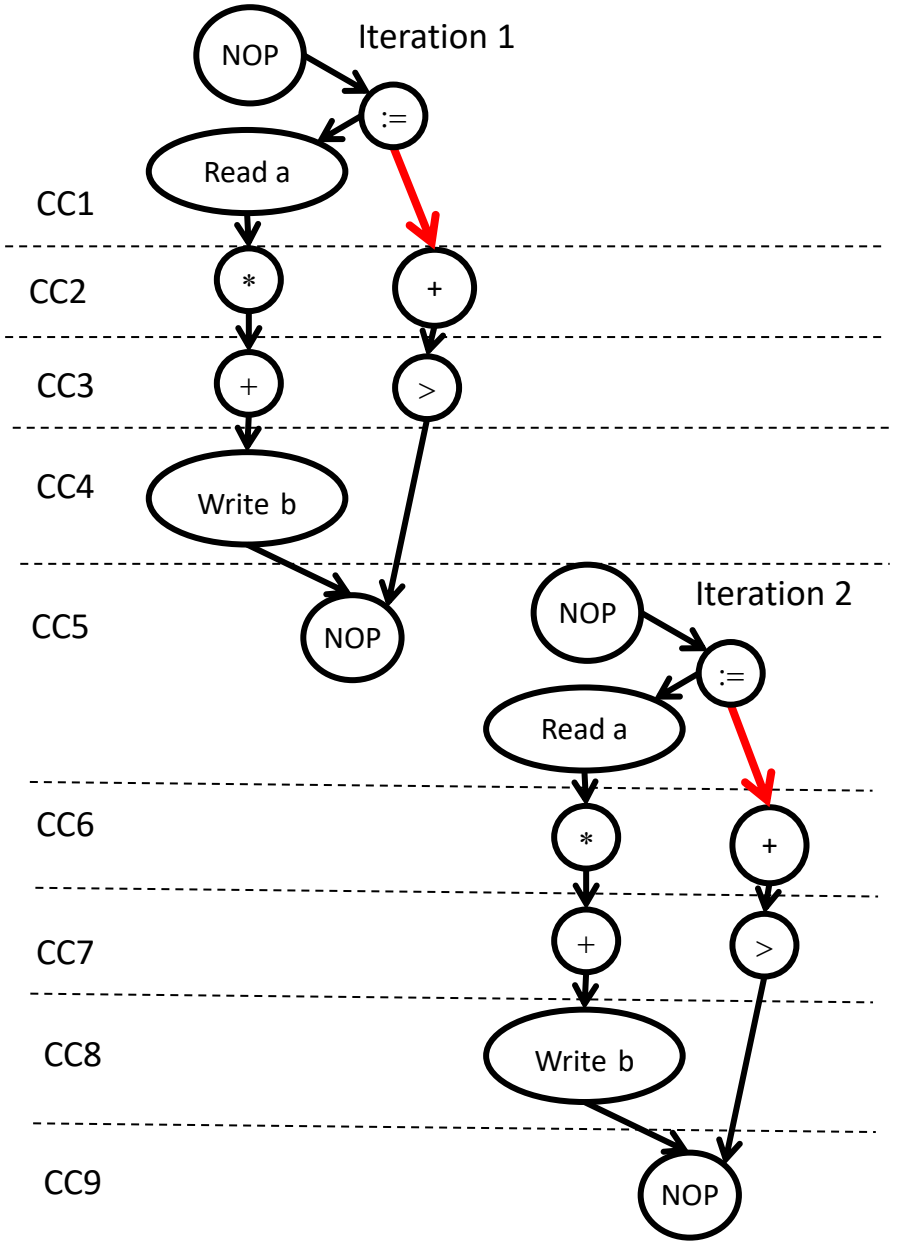


Sequential Loop Schedule

- Loop iterations scheduled after each other
- 4 cycles*20=80 cycles

Three-address code

```
i:=0
B1:t1:=i   //THR
   t2:=a[t1]
   t3:=t2*t2
   t4:=t2+t3
   b[t1]:=t4
   i=i+1
   if (i<20) goto B1
```



Full Loop Unrolling

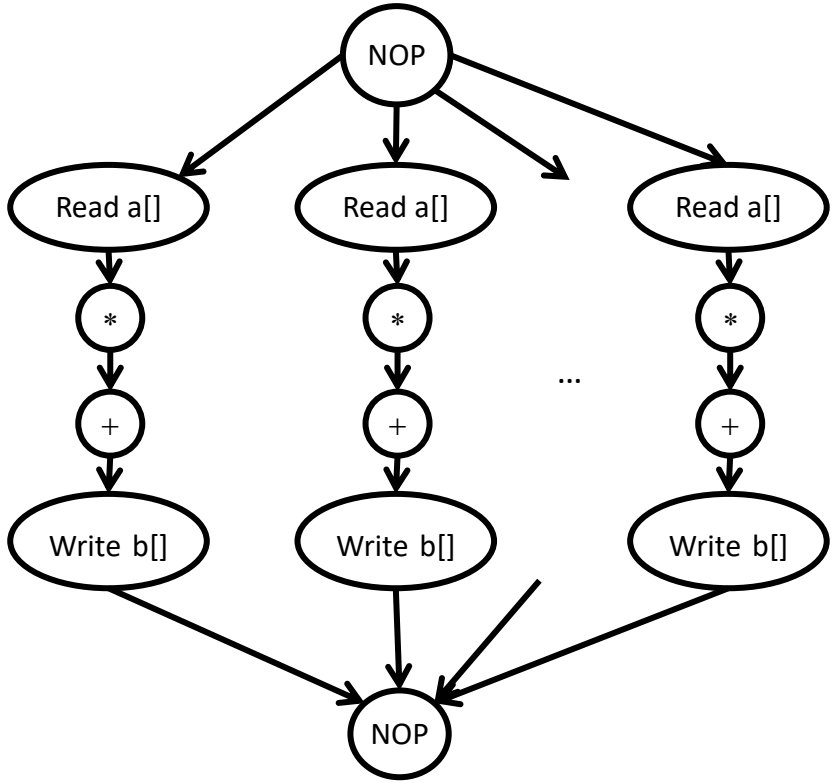
- Loop iterator and condition removed: Full parallism
- Requires many resources.
- Scheduling with known methods.
- 4 cycles

C-Code section

```
void loopex(int a[20],int b[20])
{
        b[0]=a[0]*a[0]+a[0];
        b[1]=a[1]*a[1]+a[0];
        …
        b[19]=a[19]*a[19]+a[19];
}
```

Three-address code

```
t1:=a[0]
t2:=t1*t1
t3:=t1+t2
b[0]:=t3
t4:=a[1]
t5:=t4*t4
…
```

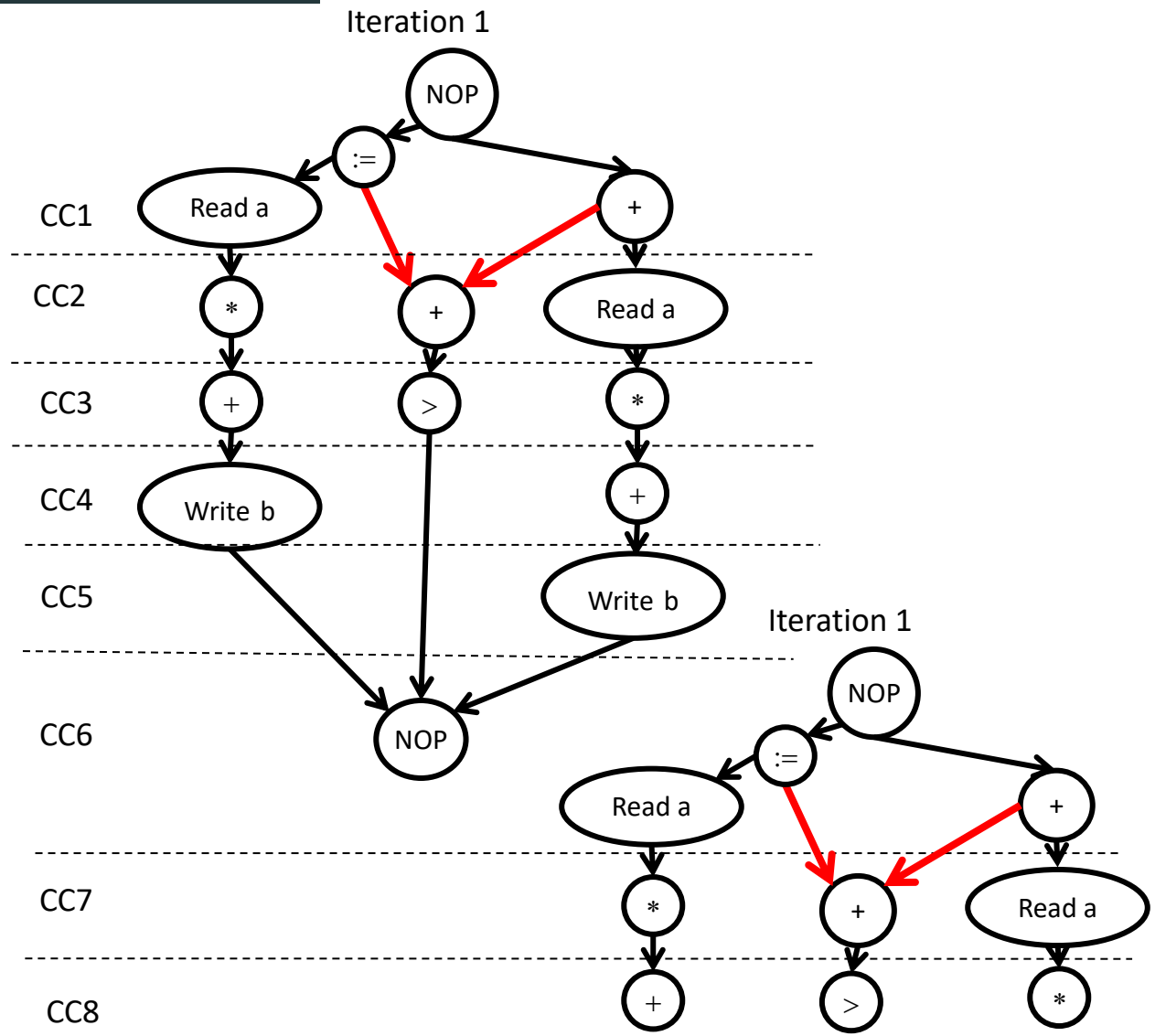


Partial Unrolled Loops

- Unroll factor = 2
- Two loop iterations scheduled in parallel
- 5 cycles * 10 = 50 cycles

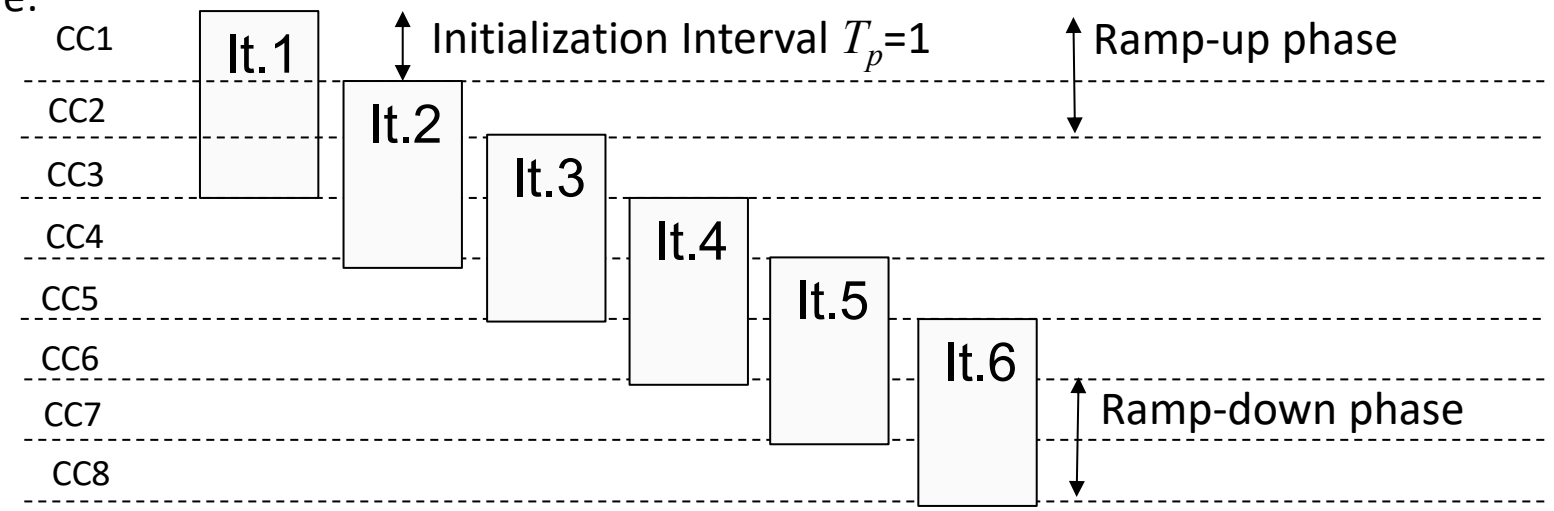
Three-address code

```
i:=0
B1:t1:=i     //THR
   t2:=i+1   //THR
   t3:=a[t1]
   t4:=t3*t3
   t5:=t4+t3
   b[t1]:=t5
   t6:=a[t2]
   t7:=t6*t6
   t8:=t7+t6
   b[t2]:=t8
   i=i+2
   if (i<20) goto B1
```



Loop Pipelining

- Execution of loop iteration starts before last loop iteration ended
- Initialization Interval T_p is delay between start of iterations
- For loop pipelining $T_p < \Lambda_{SGU,LOOP}$
- Start time of nodes for different iterations $k,k+1$: $t_i^{(k+1)} = t_i^{(k)} + T_p$
- Latency of Loop Node $d_{Loop} = T_p \cdot \#iterations + (\Lambda_{SGU,LOOP} - T_p)$
- Example:

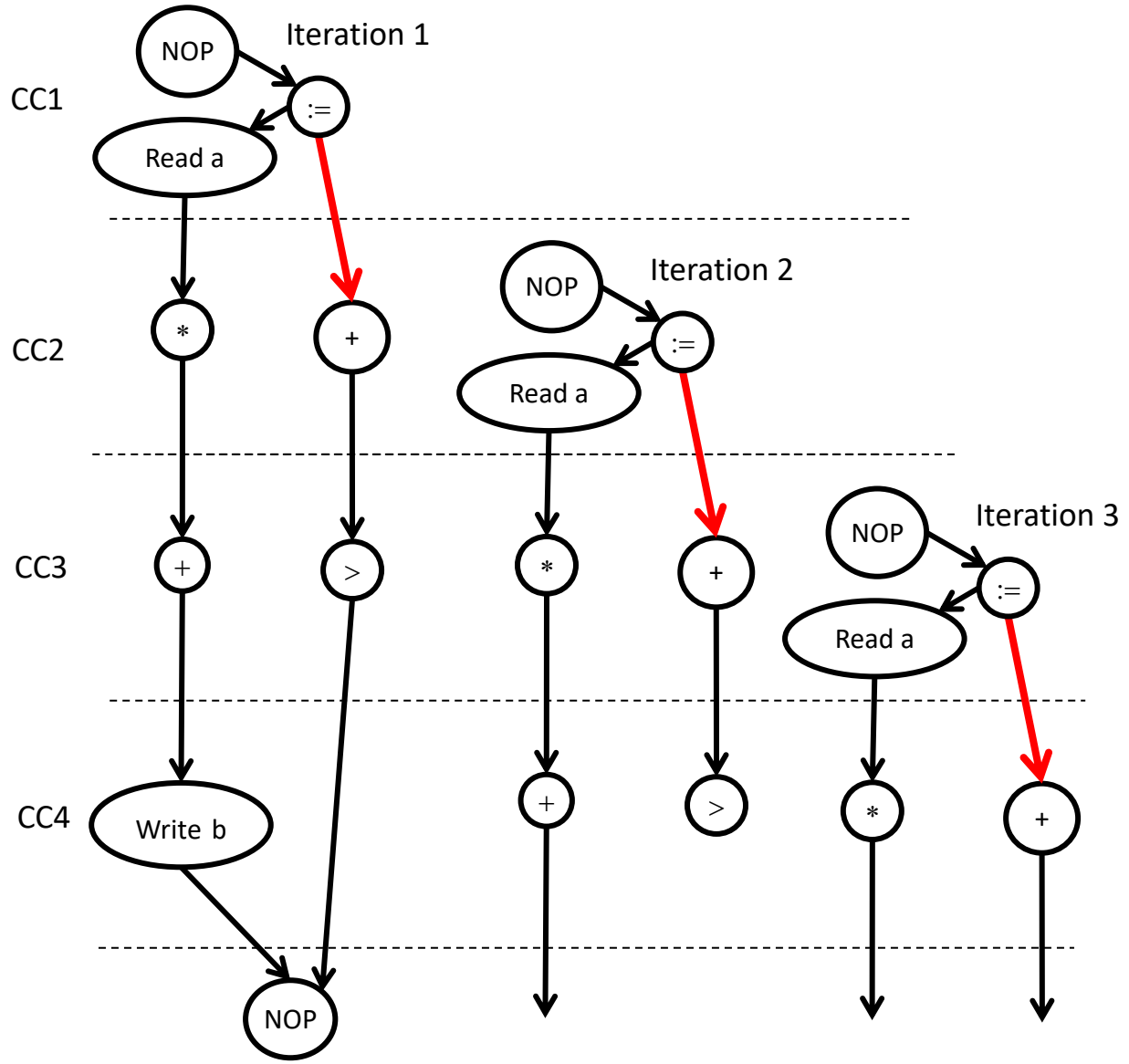


Pipelined Loop Schedule

- Loop iterations scheduled with T_p=1
- 1 cycle * 20 + 4 cycles – 1 cycle = 23 cycles

Three-address code

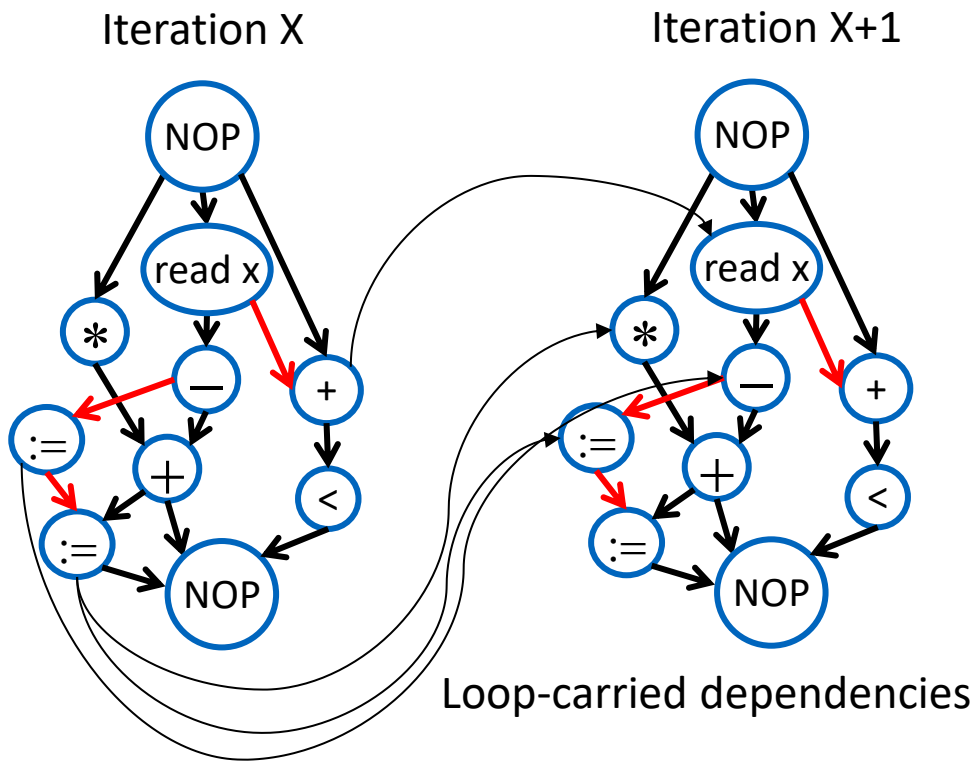
```
i:=0
B1:t1:=i   //THR
   t2:=a[t1]
   t3:=t2*t2
   t4:=t2+t3
   b[t1]:=t4
   i=i+1
   if (i<20) goto B1
```



Scheduling Do-accross Loops

- Loop-carried dependencies: Data dependencies between loop iterations
- Example: Goertzel Algorithmus

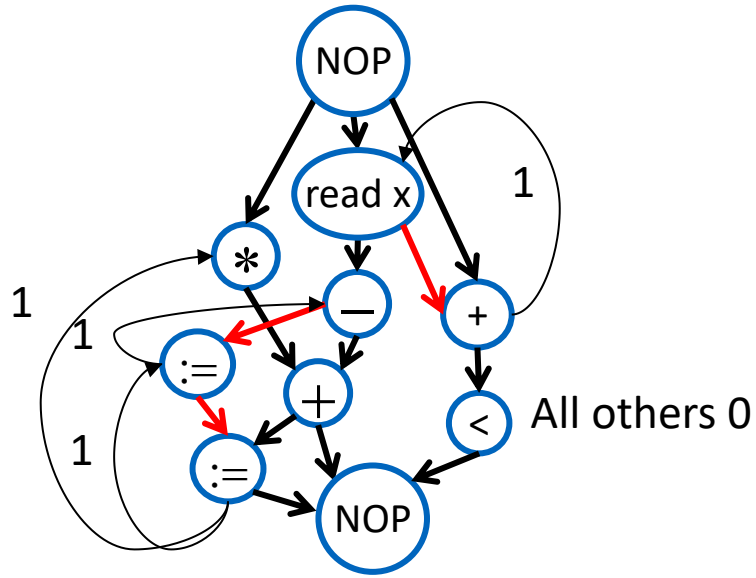
```
B2: t3:= coeff * s_prev1
    t4:= x[i]
    t5 := t4 - s_prev2
    s := t3 + t5
    s_prev2 := s_prev1
    s_prev1 := s
    i:=i+1
    if i < 64 goto B2
```



SGU with Loop-Carried Dependencies

- $G_s(V_s,E_s)$
- $Es=\{e_k=(v_i,v_j):\ i,j=K\ldots S\}$
- Edge weight: $\delta(e_k) = \delta\left((v_i, v_j)\right) = w_k \quad \delta: E_s \rightarrow \mathbb{Z}^* = \{0,1,2..\}$
  - w_K: Number of iterations the loop-carried dependency crosses
  - $w_K=0$: Dep. In same iteratinos (as in standard SGU)
- SGU not an directed acyclic graph (DAG) anymore
- Example: Goertzel Algorithmus

```
B2: t3:= coeff * s_prev1
    t4:= x[i]
    t5 := t4 - s_prev2
    s := t3 + t5
    s_prev2 := s_prev1
    s_prev1 := s
    i:=i+1
    if i < 64 goto B2
```



Constraints for Loop-Carried Dependencies

- Loop carry-dependencies lead to additional constraints

For $e_k=(v_i,v_j)$ with $w_k>0$: $t_j^{(l+w_k)} \geq t_i^{(l)} + d_i$

Operation j in iteration $(l+wk)$ may only start after operation i in iteration (l) finished.

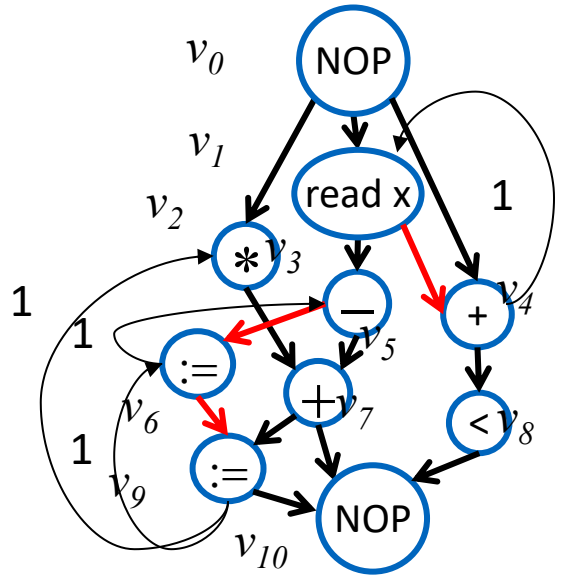
- Example: Goertzel Algorithmus

$$t_2^{(l+1)} \geq t_9^{(l)} + d_9$$

$$t_5^{(l+1)} \geq t_6^{(l)} + d_6$$

$$t_6^{(l+1)} \geq t_9^{(l)} + d_9$$

$$t_1^{(l+1)} \geq t_4^{(l)} + d_4$$



Pipelining do-across loops

- Constraints for pipelining

For $e_k=(v_i,v_j)$ with $w_k>0$: $t_j^{(l+w_k)} \geq t_i^{(l)} + d_i$

For pipelining: $t_j^{(l+w_k)} = t_j^{(l)} + w_k \cdot T_p$

Constraint for loop-carry dep: $t_j^{(l)} + w_k \cdot T_p \geq t_i^{(l)} + d_i$

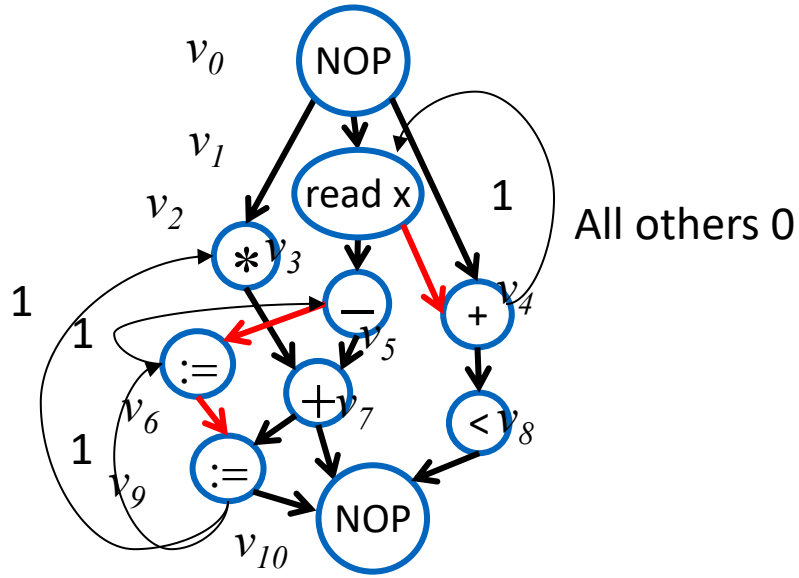
- Example: Goertzel Algorithmus

$$t_2^{(l)} + T_p \geq t_9^{(l)} + d_9$$

$$t_5^{(l)} + T_p \geq t_6^{(l)} + d_6$$

$$t_6^{(l)} + T_p \geq t_9^{(l)} + d_9$$

$$t_1^{(l)} + T_p \geq t_4^{(l)} + d_4$$



Pipelining do-across loops

- Example: Scheduled Loop of Goertzel Algorithmus

$$t_2^{(l)} + T_p \geq t_9^{(l)} + d_9$$

$$t_5^{(l)} + T_p \geq t_6^{(l)} + d_6$$

$$t_6^{(l)} + T_p \geq t_9^{(l)} + d_9$$

$$t_1^{(l)} + T_p \geq t_4^{(l)} + d_4$$

Assignment with zero execution time $d_6=d_9=0$

$$2 + T_p \geq 4 \Rightarrow T_p \geq 2$$

$$2 + T_p \geq 3$$

$$3 + T_p \geq 4$$

$$1 + T_p \geq 3 \Rightarrow T_p \geq 2$$

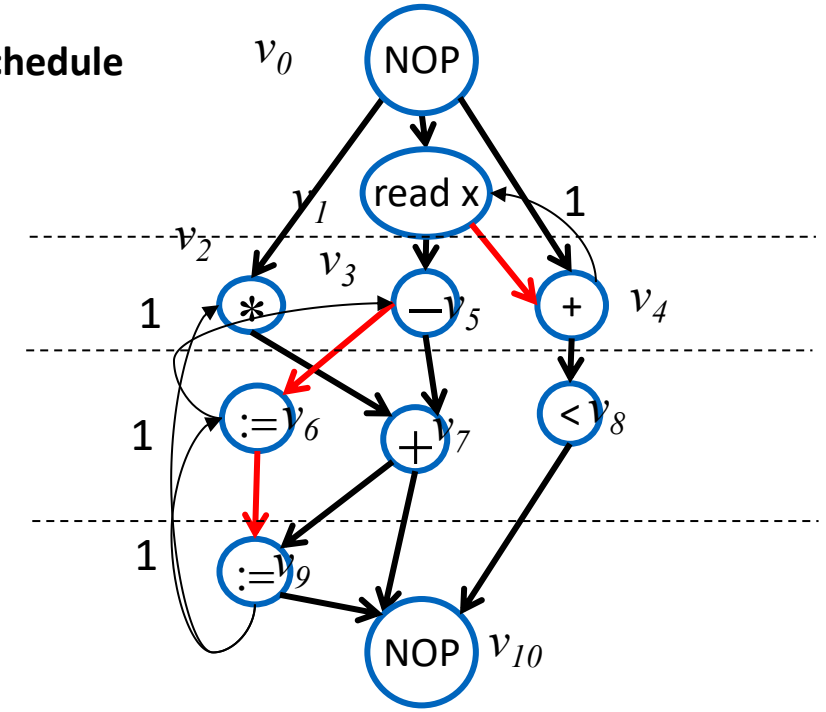
Pipelining with $T_p=2$ possible

ALAP Schedule

$t_2=t_4=t_5=2$

$t_6=t_9=3$

$t_9=4$



Scheduling needs to find solution fulfilling all constraints and possible also resource constraints. List scheduling not possible due to cycles!
SOLUTION: Scheduling with Integer Linear Programming (ILP)

Loop Implementation with Counter

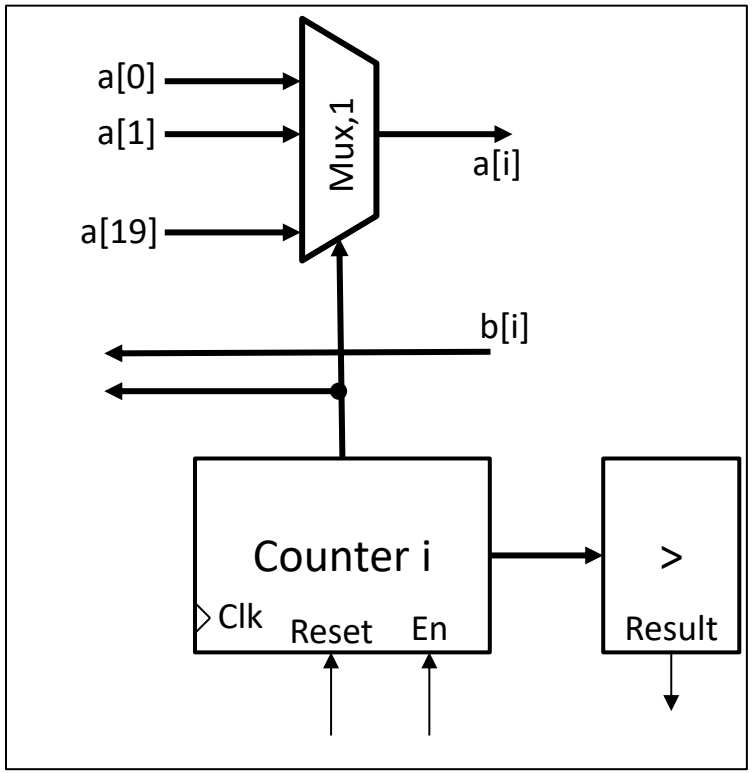
- Loop iterator can be implemented as counter (Operation: Acc)
- Loop condition can be implemented as comparator (Operation: >)

C-Code section

```
void loopex(int a[20],int b[20])
{
     for (int i=0;i<20;i++)
          b[i]=a[i]*a[i]+a[i];
}
```

Three-address code

```
i:=0
B1:t1:=a[i]
   t2:=t1*t1
   t3:=t1+t2
   b[i]:=t3
   i=i+1
   if (i<20) goto B1
```



Loop Implementation with Counter

- Doubling of interfaces for loop unrolling with factor 2

C-Code section (Unroll factor 2)

```
void loopex(int a[20],int b[20])
{
  for (int i=0;i<20;i=+2) {
    b[i]=a[i]*a[i]+a[i];
    b[i+1]= a[i+1]*a[i+1]+a[i+1];
  }
}
```

Three-address code

```
i:=0
B1:t1:=i
   t2:=i+1
   t3:=a[t1]
   t4:=t3*t3
   t5:=t4+t3
   b[t1]:=t5
   t6:=a[t2]
   t7:=t6*t6
   t8:=t7+t6
   b[t2]:=t8
   i=i+1
   if (i<20) goto B1
```