Chapter 3 Annex – VLIW Examples

VLIW Architecture

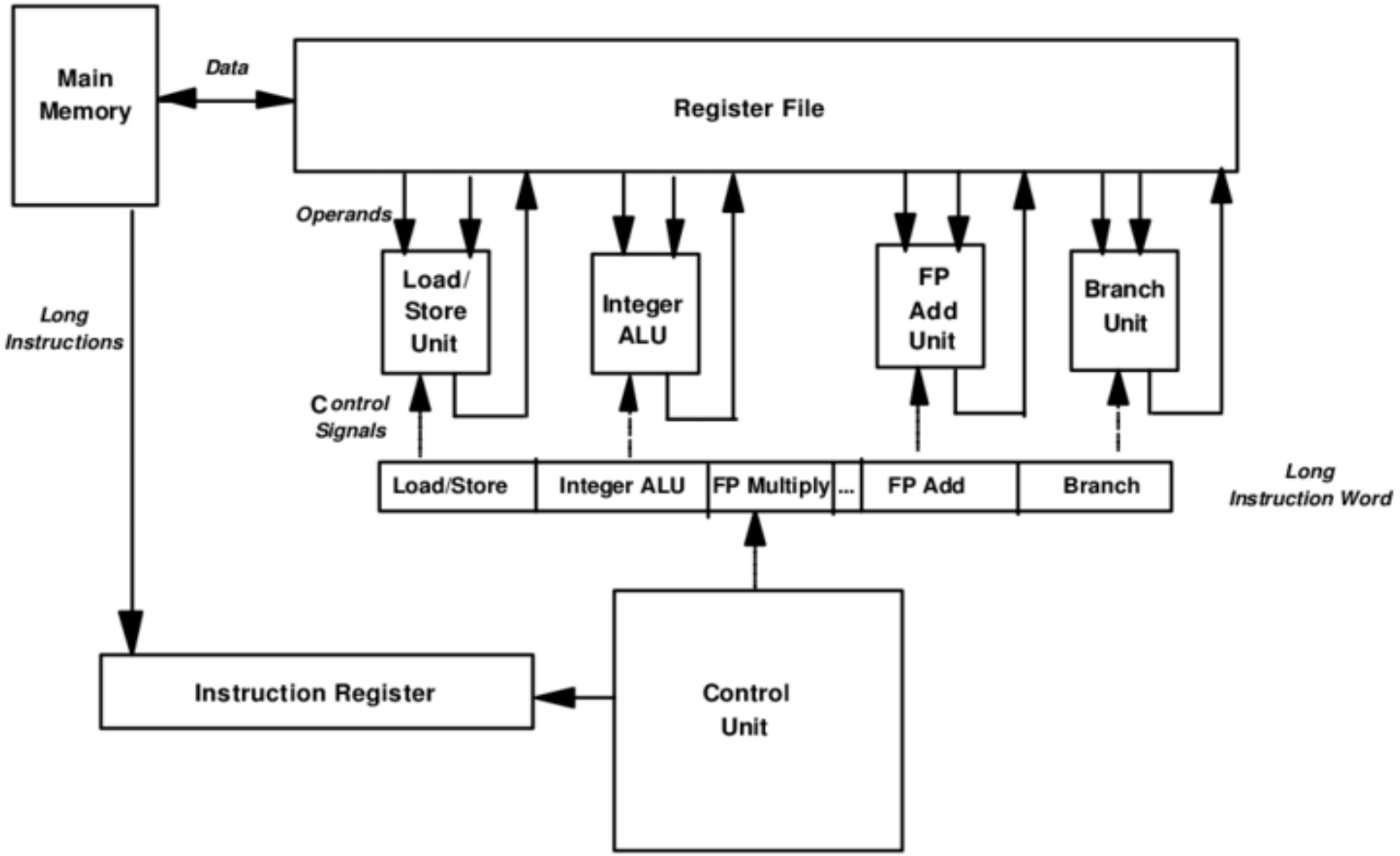
- Generalization of two concepts: *horizontal micro-coding* and *superscalar processing*

A VLIW machine has instruction words hundreds of bits in length. (*See Fig. next slide*)

Multiple functional units are used concurrently and all these units share a common register file. *Those operations that are required to be executed simultaneously by the functional units are synchronized in a VLIW instruction, say, 256 or 1024 bits per inst. word.*

Pipelining in VLIW machines

Here, each instruction specifies multiple operations.



Example 1 (ILP in VLIW)

Computation: $y = a_1*x_1 + a_2*x_2 + a_3*x_3$

Seq Processor:

Load a1
Load x1
Load a2
Load x2
MUL z1,a1,x1
MUL z2,a2,x2
ADD y,z1,z2
Load a3
Load x3
MUL z1,a2,x3
ADD y,y,z2

VLIW Processor (2 load/store units, 1 MUL, 1ADD)

Load a1 Load x1	Load a2 Load x2 MUL z1,a1,x1	Load a3 Load x3 MUL z2,a2,x2
MUL z3,a3,x3 ADD y,z1,z2		
ADD y,y,z3		

Seq Proc: 11 Cycles
VLIW: 5 Cycles

Example 2 (Handling loops in VLIW)

Consider the following code in C/C++:

```
float arr[1000], c;
...
for (i=999; i > 0; i--)
  arr[i] = arr[i] + c;
```

First read this carefully:

Floating point numbers - 8 bytes
F**k** – **k**-th Floating point register
Assume:
F7: Holds the constant c
R1 contains the address of the last element of the array (999)
F11 - After adding the array value and the constant the result is put back in F11

Example 2 (Handling loops in VLIW)… cont'd

Delay specifications common for sequential processor and VLIW:

- Two memory references, two FP operations, and one integer operation or branch can be issued each clock cycle.
- The delay for a double word load is one additional clock cycle.
- The delay for a floating point operation is two additional clock cycles.
- No additional clock cycles for integer operations.

Example 2 (Handling loops in VLIW)… cont'd

Sequential Code:

Come_Here: *LDD F1, (R1) F1 ← x[i] ;(load float # operation)*
ADF F11, F1, F7
F11 ← F1 + F7 ;(float addition)
STD (R1), F11 (x[i] ← F11; store float # operation)
SBI R1,R1,#8 R1 ← R1 - 8
BGEZ R1, Come_Here (branch greater than zero)

Note that in Cy 5 SBI is done but in Cy 6 STD is done with (8+R1); So, F11 writes into the correct x[i].

Sequential Timing Solution:

					BGEZ
LDD F1,(R1)		**ADF F11, F1, F7**		**SBI**	**STD 8(R1),F11**
1	*2*	*3*	*4*	*5*	*6*

Example 2 (Handling loops in VLIW)… cont'd

1 iteration took: 6 cycles;
*So, total time taken: 1000 * 6 = 6000 cycles*
- *No parallelism here to exploit!*
- *No action in 2 cycles (2nd and 4th cycles)*

Note that in the original loop, the data dependency is not there since only the i-th element is updated and it is not dependent on (i-1) or (i+1);

Loop unrolling must be possible (Chapter 2)!

Let us try to unroll the loop and see if we can identify parallelism and hence can make use of VLIW!

Example 2 (Handling loops in VLIW)… cont'd

```
for (i=999; i > 0; i -=2)
  arr[i] = arr[i] + c;
  arr[i-1]  = arr[i-1] +c
```

VLIW Solution via Loop Unrolling

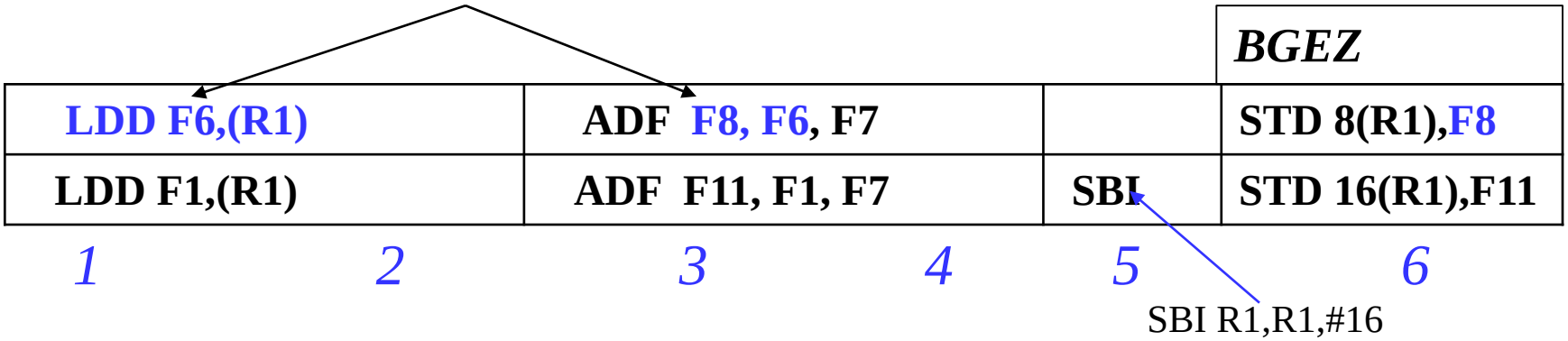
```
Come_Here: LDD F1, (R1)
           ADF F11,F1,F7
           STD (R1),F11
           LDD F1, -8(R1)
           ADF F11,F1,F7
           STD -8(R1),F4
           SBI R1,R1,#16
           BGEZ R1, Come_Here
```

Note: Following the earlier implementation, the operations -8(R1) and subtracting 16 are for memory pointer movements to access the correct data. Trace and understand.

Example 2 (Handling loops in VLIW)… cont'd

Loop unrolling - timing solution

Note: New registers are used while implementing for loop unrolling case



2 iterations in 6 cycles!

*So, total time taken in this case: 500 * 6 = 3000 cycles*

- *No parallelism here to exploit!*
- *No action in 2 cycles (2nd and 4th cycles)*

Example 2 (Handling loops in VLIW)… cont'd

```
for (i=999; i > 0; i -=3)
  arr[i] = arr[i] + c;
  arr[i-1]  = arr[i-1] +c
  arr[i-2]  = arr[i-2] +c
```

VLIW Solution via Loop Unrolling

Following the earlier implementations, we need to use 8, 16, and 24 to be subtracted carefully to perform memory pointer movements to access the correct data. *Trace and understand!*

In this case, we can arrive at an implementation that can allow 3 iterations to be completed in 7 cycles!
So, total time taken will be 333.3 * 6 ~= 2000 cycles

Continuing further, we can realize that 8 iterations would take 9 cycles and this leads to a total time of, ~= 750 cycles

Example 2 (Handling loops in VLIW)… cont'd

VLIW Solution via Loop Unrolling

Closing remarks:

Thus, it may be noted that,

- Given a certain set of resources (processor architecture, regs, memory space, etc) and a given (independent) loop, there exist a limit on how many iterations can be handled/unrolled. Beyond that limit there is no gain any more.
- So, the role of a good compiler is to determine the optimal level of unrolling for each loop.
- One of the clear disadvantages is that, the loop unrolling increases the memory space needed to store the program.