



Integrated Code Generation by Using Fuzzy Control System

Dipl.-Ing. Xiaoyan Jia

I Introduction

- STA Architecture
- Code Generation for STA

II Analysis of Problem in Code Generation

- Access of Results in STA
- Integrated Code Generation

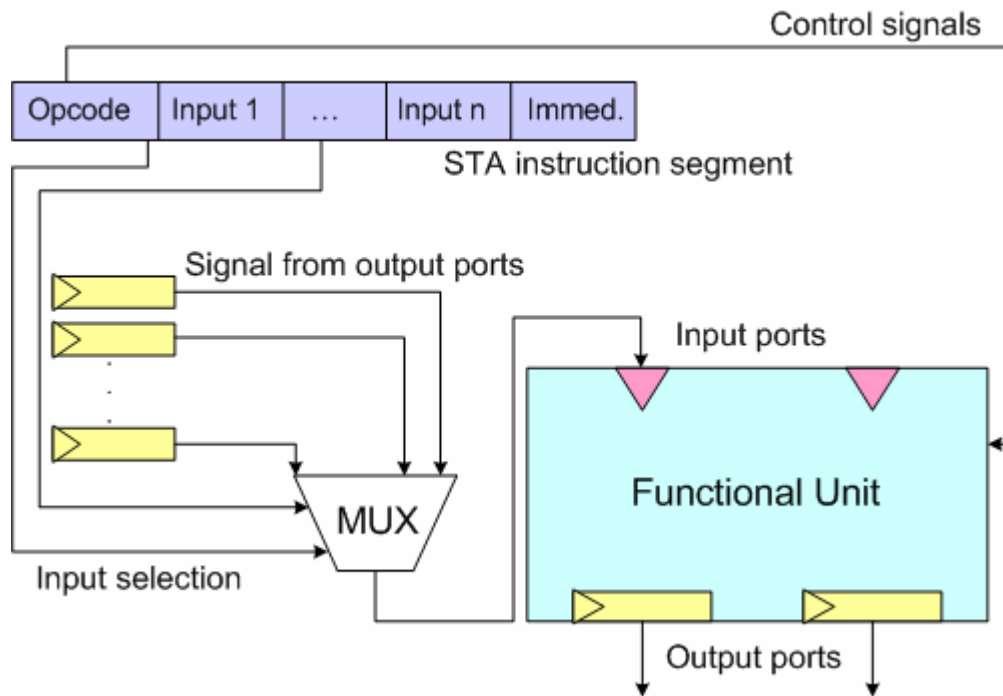
III Algorithm

- Basic Concepts in Fuzzy
- Details of Integrated Code Generation

IV Experimental Result

V Conclusion

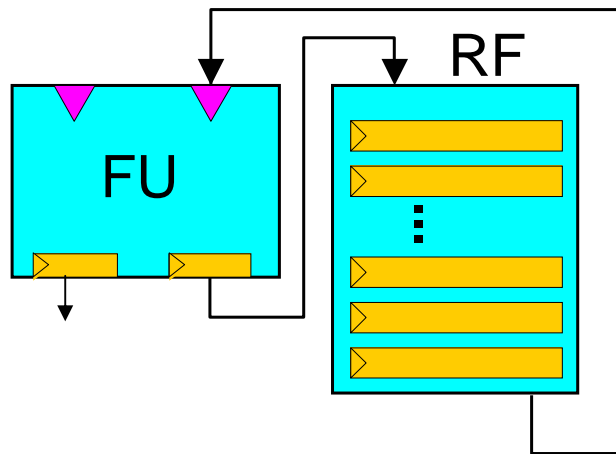
STA Architecture



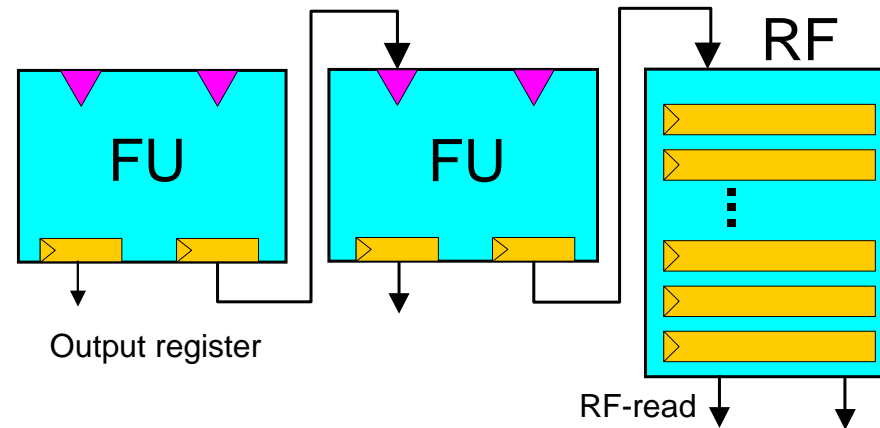
- ❖ Modules consist of input and output ports
- ❖ Output ports are buffered
- ❖ Results can be held further in output registers until the execution of the next operation is finished
- ❖ Data at an input port is selected from a set of connected output ports by a MUX (DDR)

Synchronous Transfer Architecture (STA)

STA Architecture and Code Generation



❑ RISC concept



❑ STA concept

Focus:

- ❖ To decide, the results should be transferred into general registers or be held further in output registers

Problem:

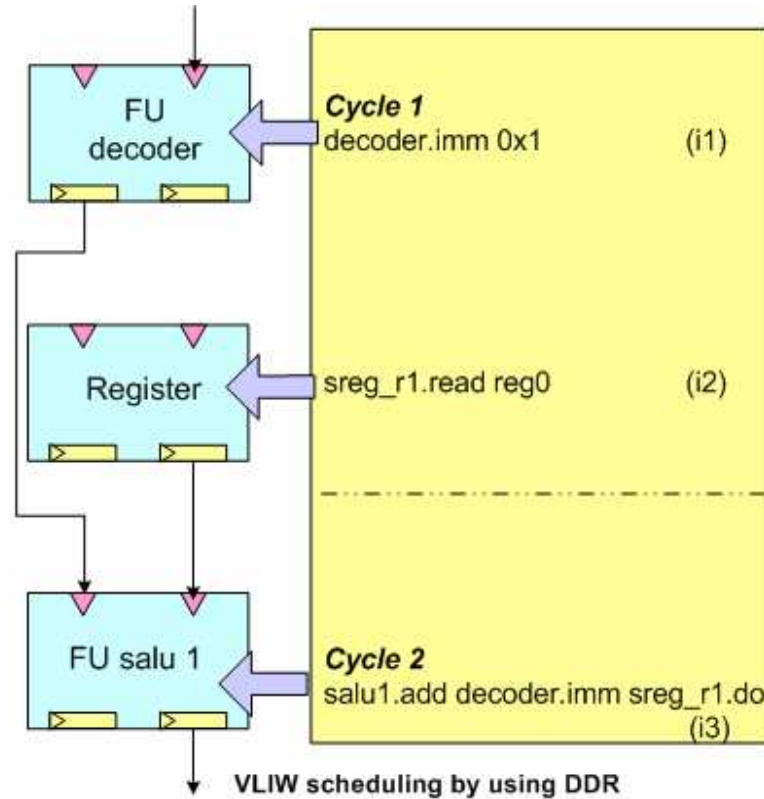
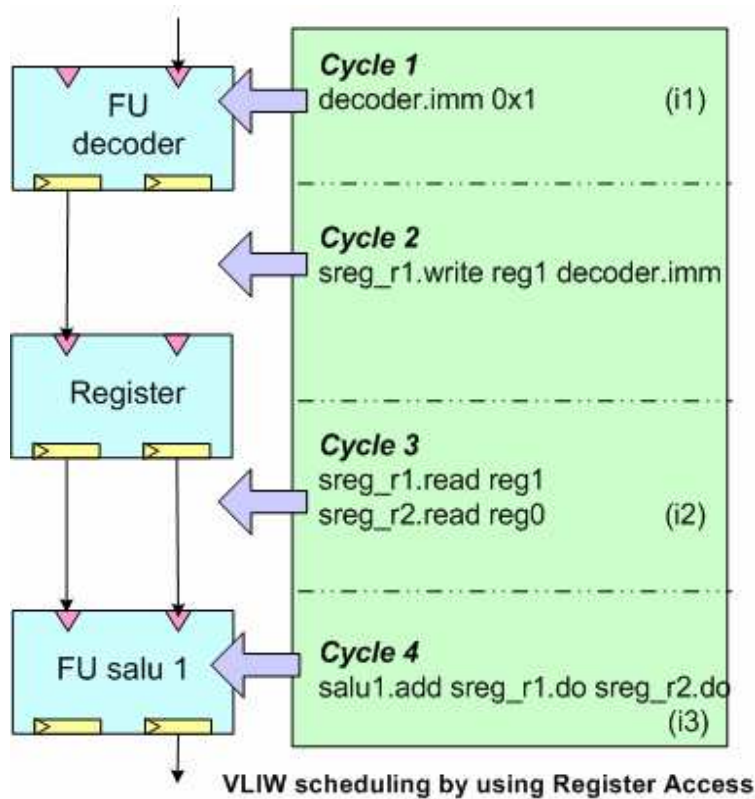
- ❖ Phase-coupling problem specially for STA architecture

Access of Results in STA

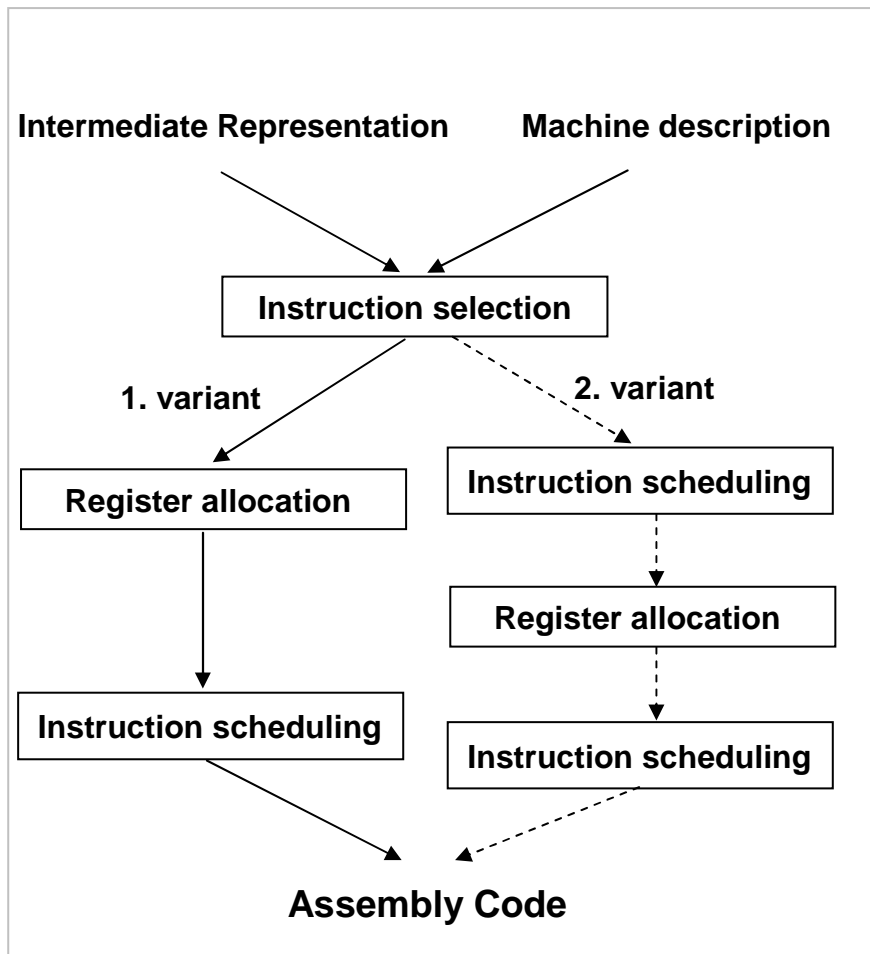
```

a = 0x1           (i1)
read b           (i2)
c = a + b        (i3)
b is stored in general register r0
    
```

Original Instructions in MIR



Phase-Coupling Problem in STA

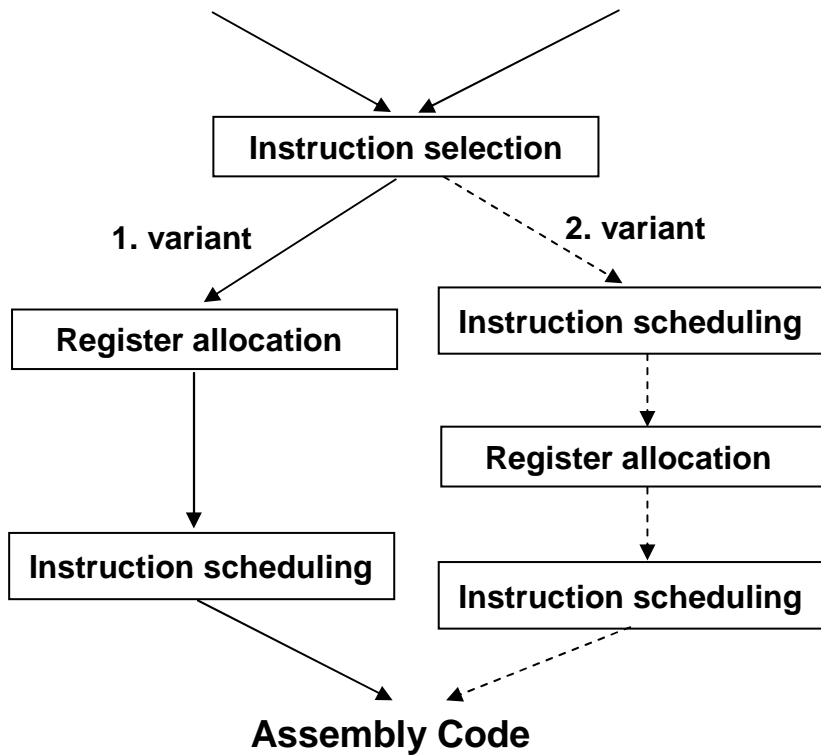


Scheduling & register allocation are separated

- ❖ **Register allocation is before scheduling**
 - Impossible to know whether DDR could be happened
 - All the results should be saved in registers
 - Poor performance and no STA feature is represented
- ❖ **Scheduling is before register allocation**
 - DDR is detected
 - FUs can not be used until results could be released
 - Reduction of instruction level parallelism

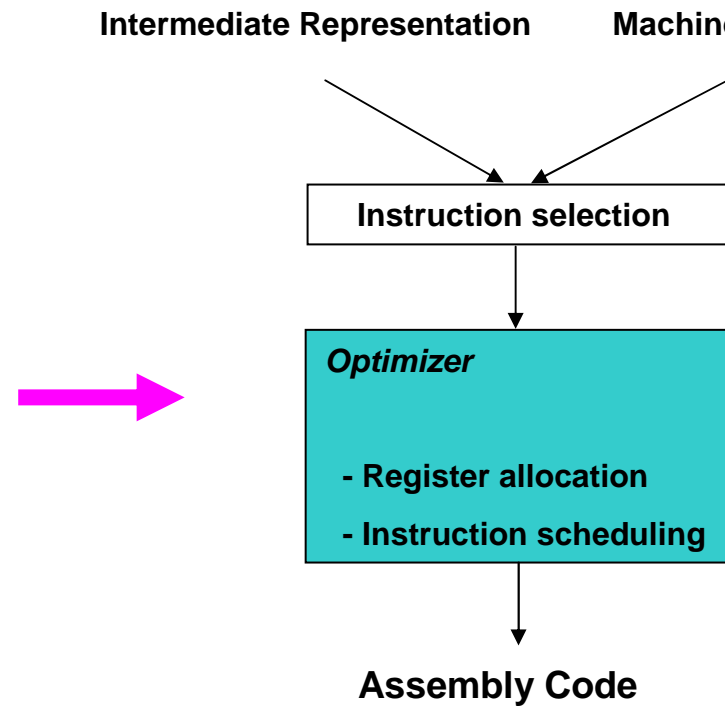
Framework of Compiler Backend

Intermediate Representation Machine Description



Traditional Compiler backend

Intermediate Representation Machine Description



Integrated Compiler backend

Fuzzy truth $\mu \in [0, 1]$ represents the membership (u) in vaguely defined sets

Fuzzy set: $A := \{(u, \mu_A(u)) \mid u \in U, \mu_A(u) \in [0, 1]\}$

Complement: $\bar{A} = \{(u, (1 - \mu_A(u))) \mid u \in U\}$

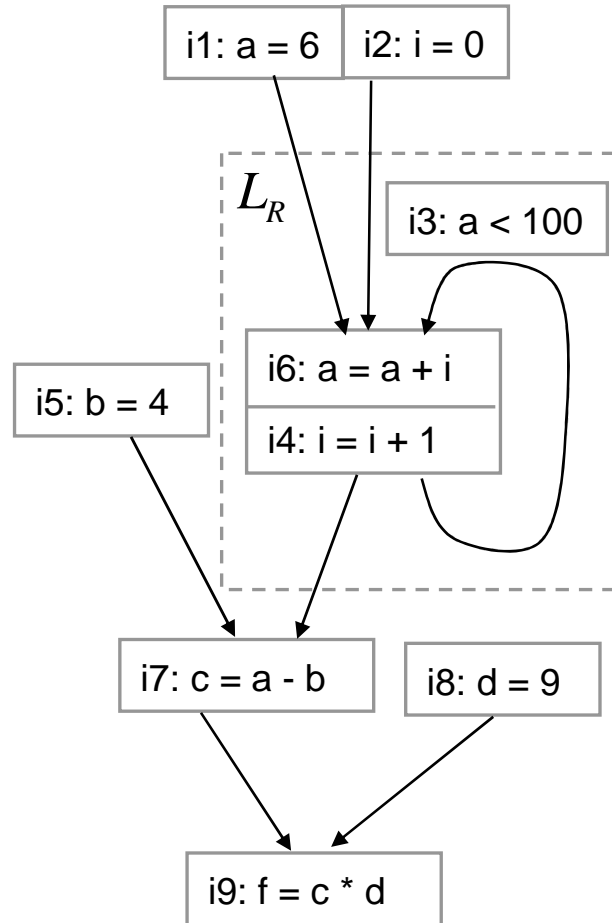
Inference: $\mu_c(u) = \mu_A(u) \cap \mu_B(u) = \min\{\mu_A(u), \mu_B(u)\} \quad \forall_u \in U$

Composition: $\mu_c(u) = \mu_A(u) \cup \mu_B(u) = \max\{\mu_A(u), \mu_B(u)\} \quad \forall_u \in U$

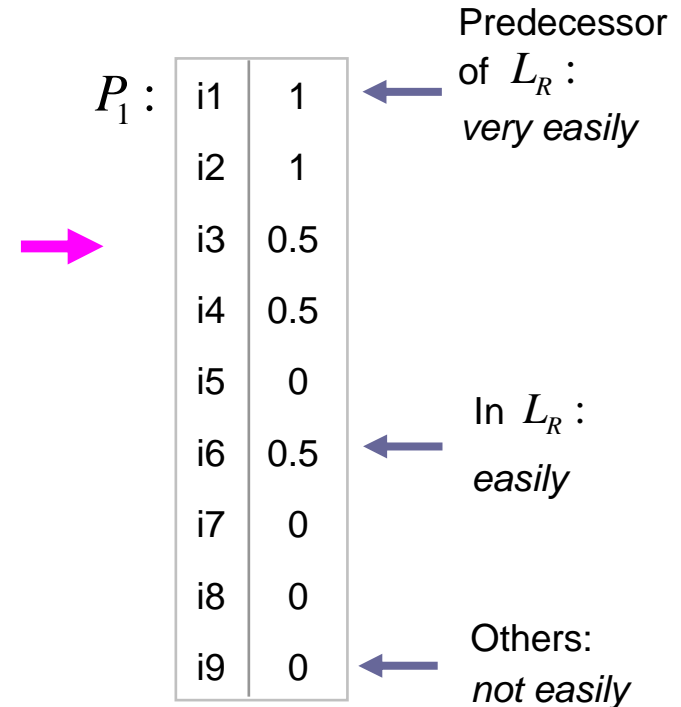
P1: Truth of Fusion into Loops

```

int a= 6;
int i, c, f;
for( i=0; a<100; i++)
{
    int b = 4;
    a = a + i;
    c = a - b;
}
int d = 9;
f = c * d;
    
```

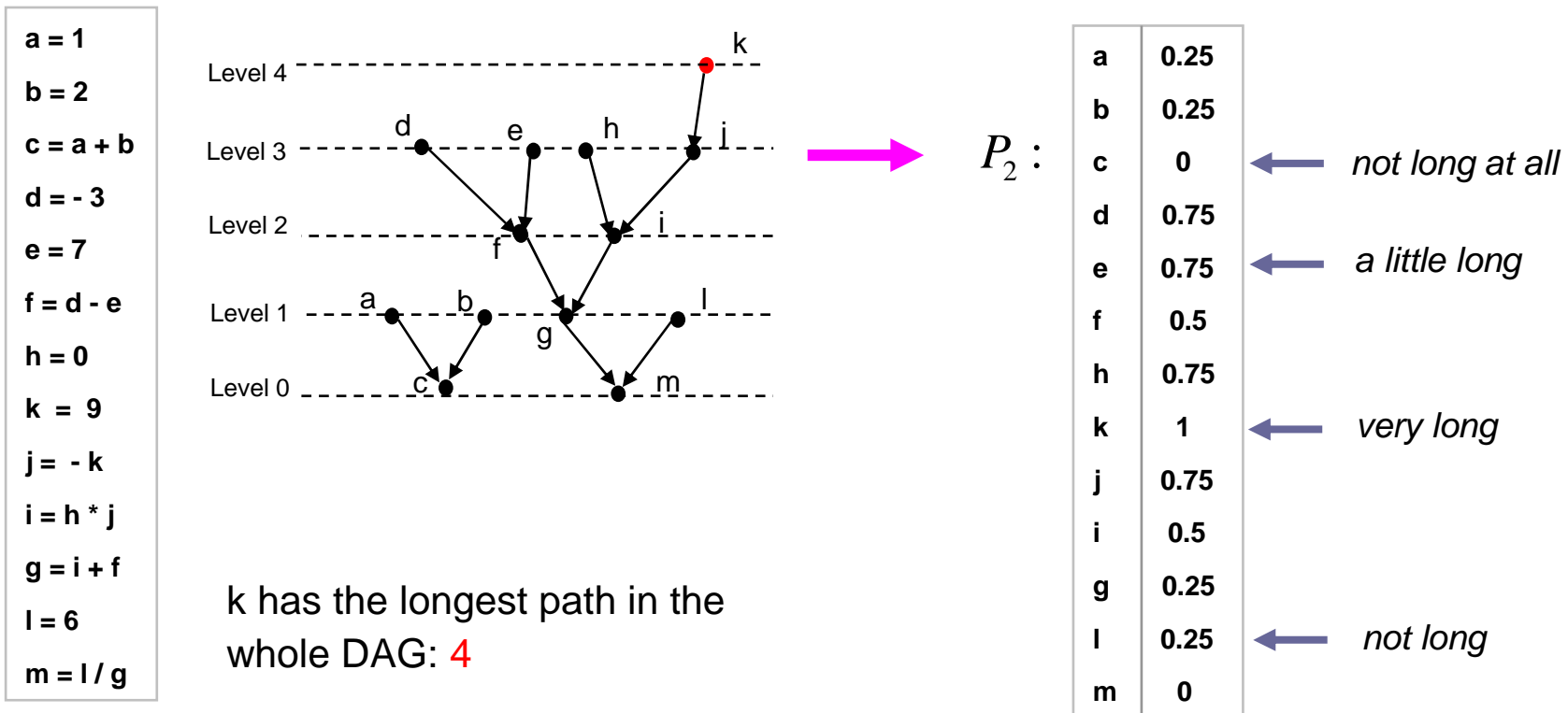


L_R : observed loop



P2: Truth of Length in DAG

The ratio of the path's length of an instruction against the longest path in the whole DAG is determined as the instruction's P_2



P3: Truth of DDR

```

read a
b = 2
c = 0
d = 3
e = a + b
f = 5
g = c + d
h = e + f
i = -g
    
```

→

```

read a
b = 2
e = a + b
    
```

P_3 :

a	0
b	0
c	0
d	0
e	0
f	0.7
g	0
h	1
i	0

← possibly

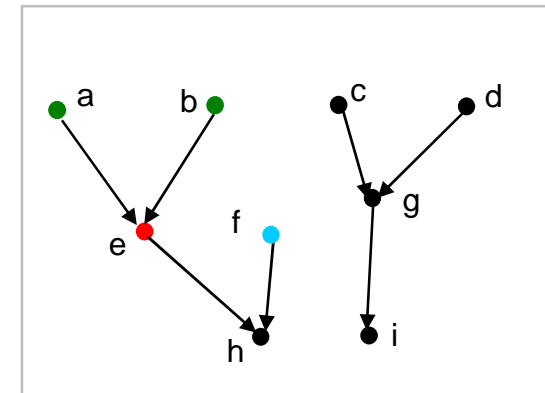
← probably

indirect DDR:

```

read a
b = 2
e = a + b
    
```

⋮
f = 5

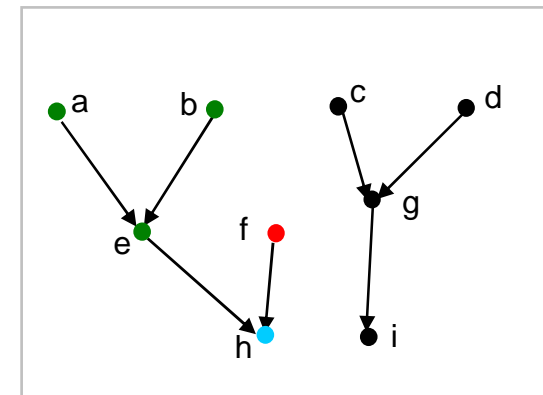


direct DDR:

```

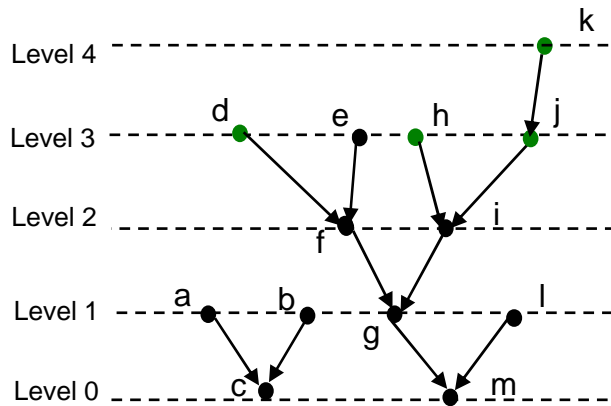
read a
b = 2
e = a + b
f = 5
    
```

⋮
h = e + f



P4: Truth of Instruction Scheduling

a = 1
b = 2
c = a + b
d = -3
e = 7
f = d - e
h = 0
k = 9
j = -k
i = h * j
g = i + f
l = 6
m = l / g



$P_4 :$

a	1
b	1
c	0
d	0
e	1
f	0
h	0
k	0
j	0
i	1
g	0
l	1
m	0

← *already scheduled*

← *with unscheduled predecessor*

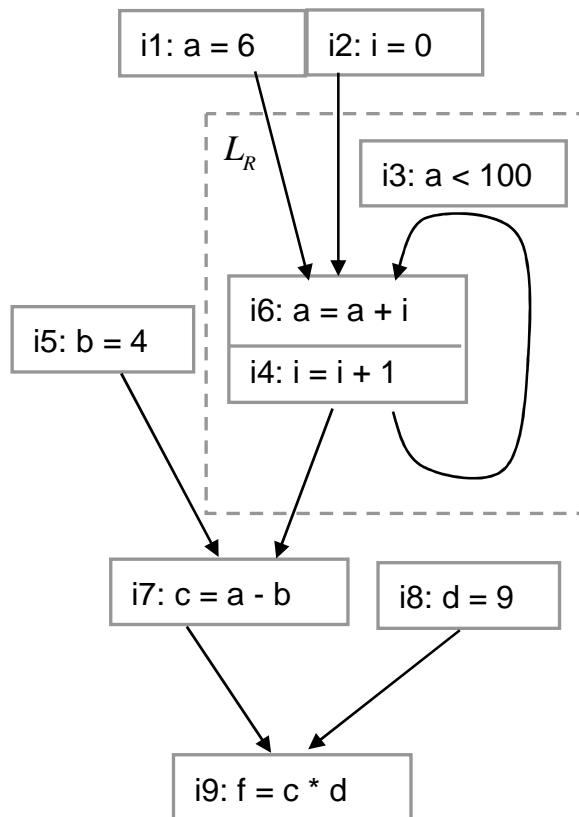
← *predecessors are all scheduled*

← *without predecessors*

Pd: Final Dynamic Factor

$$P_d = P_1 \cap (P_2 \cup P_3) \cap P_4 = \min(P_1, \max(P_2, P_3), P_4)$$

$$P_1, P_2, P_3, P_4 \in [0, 1]$$



	P ₁	P ₂	P ₃	P ₄	P _d
i1	1	1	0	1	1
i2	1	1	0	1	1
i3	0.5	1	0	0	0
i4	0.5	2/3	0	0	0
i5	0	2/3	0	1	0
i6	0.5	2/3	0	0	0
i7	0	1/3	0	0	0
i8	0	1/3	0	1	0
i9	0	0	0	0	0

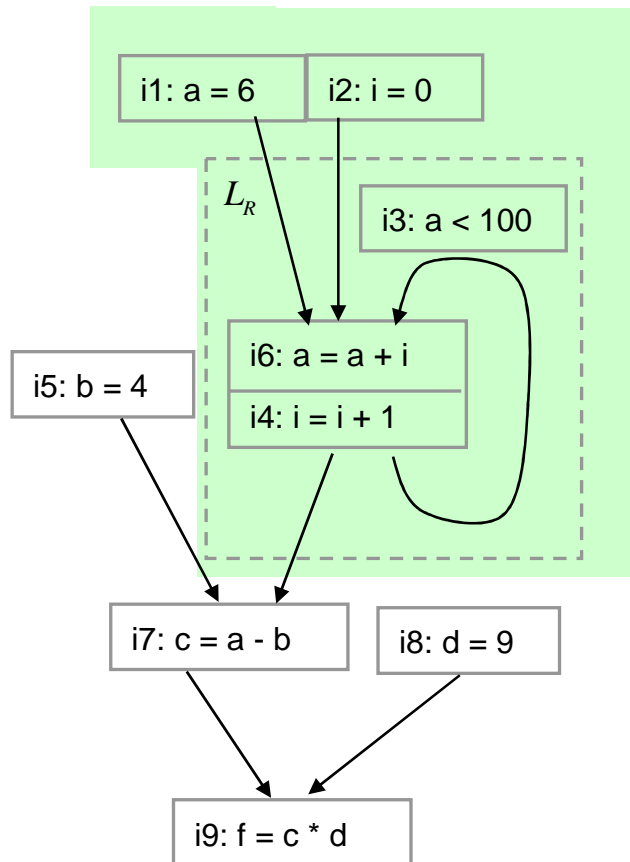


	P ₁	P ₂	P ₃	P ₄	P _d
i1	1	1	0	0	0
i2	1	1	0	0	0
i3	0.5	1	0	0	0
i4	0.5	2/3	1	0	0
i5	0	2/3	0	1	0
i6	0.5	2/3	1	1	0.5
i7	0	1/3	0	0	0
i8	0	1/3	0	1	0
i9	0	0	0	0	0

Pd: Final Dynamic Factor

$$P_d = P_1 \cap (P_2 \cup P_3) \cap P_4 = \min(P_1, \max(P_2, P_3), P_4)$$

$$P_1, P_2, P_3, P_4 \in [0, 1]$$



	P ₁	P ₂	P ₃	P ₄	P _d
i1	1	1	0	0	0
i2	1	1	0	0	0
i3	1	1	0	0	0
i4	1	2/3	0	0	0
i5	1	2/3	0.7	1	0.7
i6	1	2/3	0	0	0
i7	1	1/3	1	0	0
i8	1	1/3	0	1	1/3
i9	1	0	0	0	0

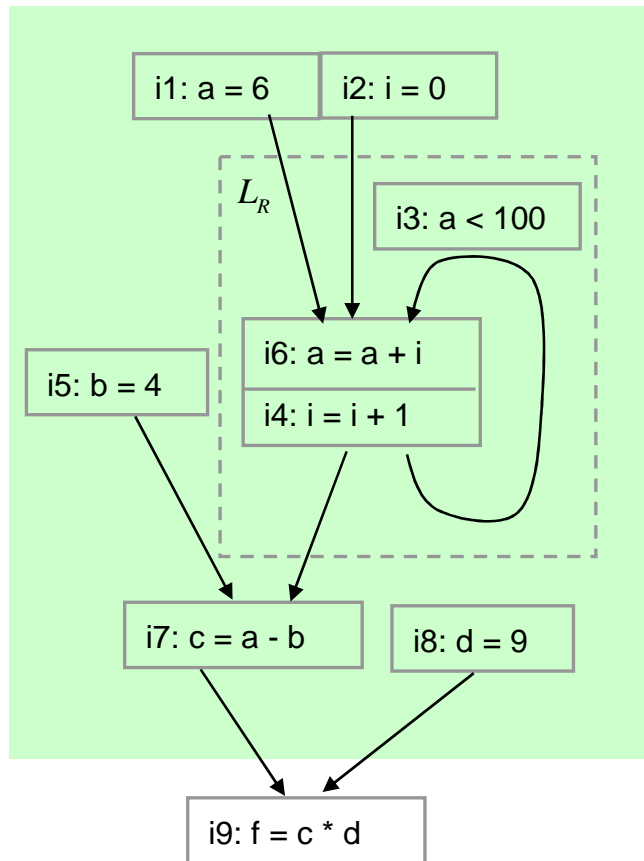


	P ₁	P ₂	P ₃	P ₄	P _d
i1	1	1	0	0	0
i2	1	1	0	0	0
i3	1	1	0	0	0
i4	1	2/3	0	0	0
i5	1	2/3	0	0	0
i6	1	2/3	0	0	0
i7	1	1/3	1	1	1
i8	1	1/3	0	1	1/3
i9	1	0	0	0	0

Pd: Final Dynamic Factor

$$P_d = P_1 \cap (P_2 \cup P_3) \cap P_4 = \min(P_1, \max(P_2, P_3), P_4)$$

$$P_1, P_2, P_3, P_4 \in [0, 1]$$



	P ₁	P ₂	P ₃	P ₄	P _d
i1	1	1	0	0	0
i2	1	1	0	0	0
i3	1	1	0	0	0
i4	1	2/3	0	0	0
i5	1	2/3	0	0	0
i6	1	2/3	0	0	0
i7	1	1/3	0	0	0
i8	1	1/3	0	0	0
i9	1	0	1	1	1



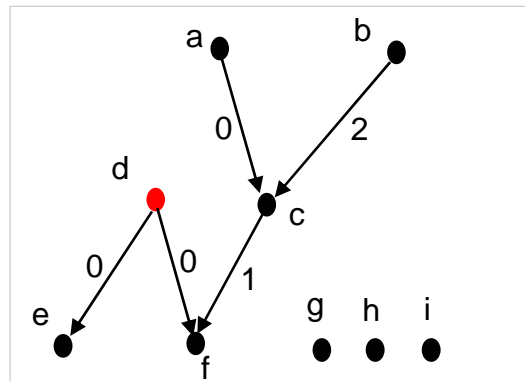
```

a = 6;
i = 0;
L1:
if (a < 100)
{
a = a + i;
i = i + 1;
goto L1;
}
b = 4;
c = a - b;
d = 9;
f = c * d;
  
```

Register Allocation

Knapsack Algorithm :

```
sld a
b = 5
c = a + b
d = - 1
e = - d
f = c * d
g = 1
h = 3
i = 5
```



For e : output register : 0

general-purpose register : write + read = 1 + 1 = 2

For f : output register : $\max(a,b) + c = 2 + 1 = 3$

general-purpose register :

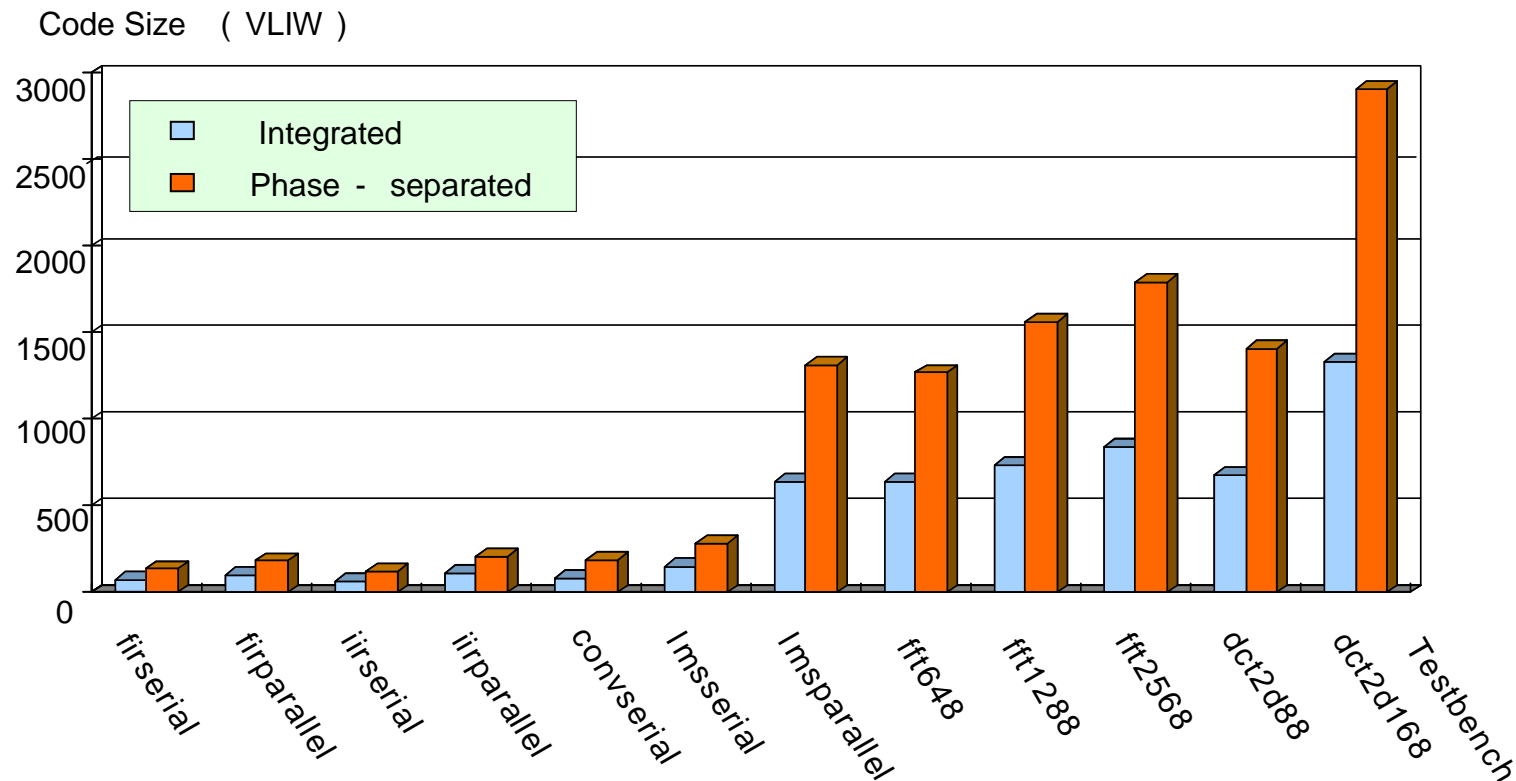
$\max((\max(a,b)+c), (\text{write} + \text{read})) = 2 + 1 = 3$

General-purpose register is **more free** than FU decoder

```
d = - 1
...
e = - d
write d
...
...
read d
f = c * d
...
```

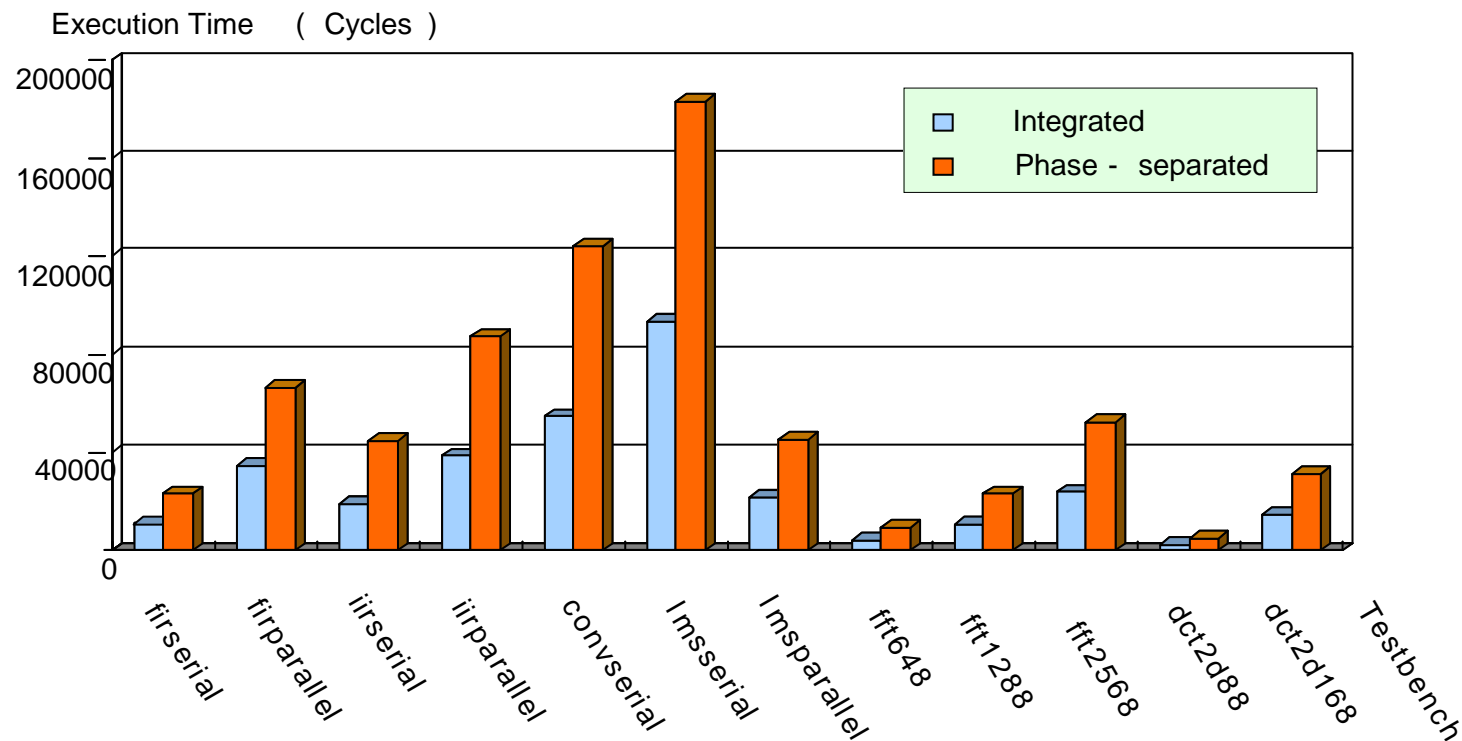
Experimental Results: *Sizes of Assembly Code*

- ❖ A **42.7% to 62.5%** code size is reduced by using integrated method in compared with the phaseseparated work



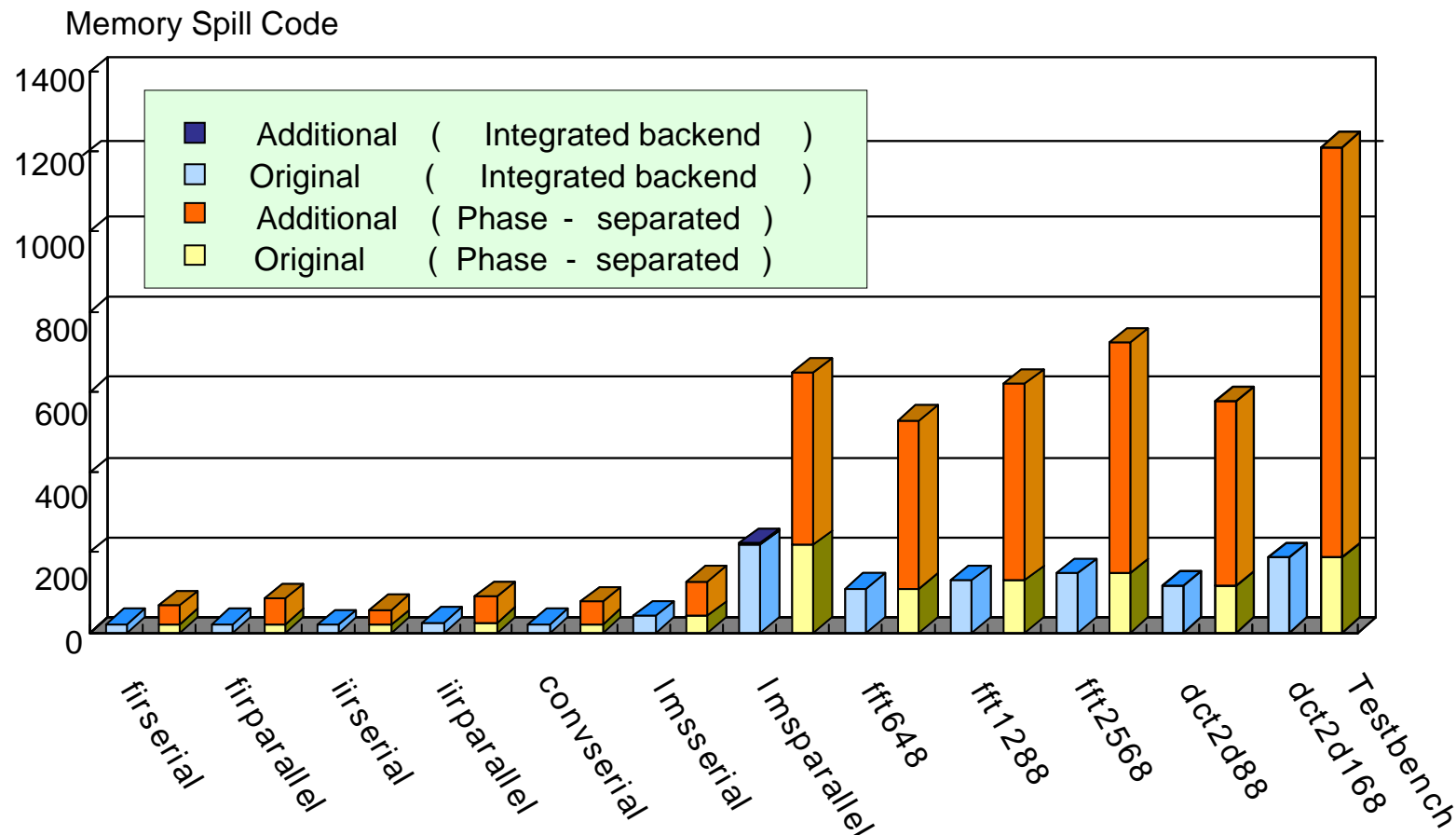
Experimental Results: *Execution Time*

- ❖ Comparison of the one by phase-separated method, the gain of integrated backend is from **45.9% to 58.3%**



Experimental Results: *Memory Spill Code*

- ❖ **Almost no** additional memory spill code are generated by using our integrated compiler backend



- ❖ The backend by using fuzzy control system and the based on ILP have ***almost the same*** code performances
- ❖ But the code generation by using fuzzy needs much ***shorter*** compilation time. For the previously mentioned testbenches, it takes only 1 second to about 20 minutes to generate code

- ❖ Comparison of the traditional phase-separated compiler backend
 - Take advantage of the STA features more efficiently
 - Generate assembly code with much better performance
 - Reduce memory spill codes greatly
 - Results much lower processor hardware power consumption

- ❖ Comparison of the Integer Linear Programming
 - Not so difficult to implement
 - Much shorter compilation time than ILP

Thank you for your attention!