Possible Answers for Final Exam (July 1, 2019)

Masters in Informatics and Computing Engineering (MIEIC), 3rd Year

João M. P. Cardoso
Dep. de Engenharia Informática, Faculdade de Engenharia (FEUP), Universidade do Porto, Porto, Portugal
Email: jmpc@fe.up.pt

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Group 4

Code1:

```
//in={}
i = 0;
m = A[0]
do {
    i=i+1;
    x = A[i];
    if (x > m)
        m = x;
} while(i<N);
// out = {m}
```

Data types for variables and storage Information:

Var.	Type	Storage
A	int8 A[N] (array of bytes)	Base array address stored in the stack at SP + 4
i	int i (32-bit scalar var.)	Variable stored in register r2
m	int m (32-bit scalar var.)	Variable stored in the stack at SP + 8
x	int x (32-bit scalar var.)	Variable stored in register r1

Instruction	Operation
load, Ri, Rj, C	Ri ← Mem[Rj+C]
store Rj, C, Ri	Mem[Rj+C] ← Ri
add Ri, Rj, Rk	Ri ← Rj + Rk
addi Ri, Rj, C	Ri ← Rj + C
mul Ri, Rj, Rk	Ri ← Rj * Rk
lth Ri, Rj, label1	If(Ri < Rj) goto label1
lte Ri, Rj, label1	If(Ri <= Rj) goto label1
gth Ri, Rj, label1	If(Ri > Rj) goto label1
gte Ri, Rj, label1	If(Ri >= Rj) goto label1
goto label1	jump to label1

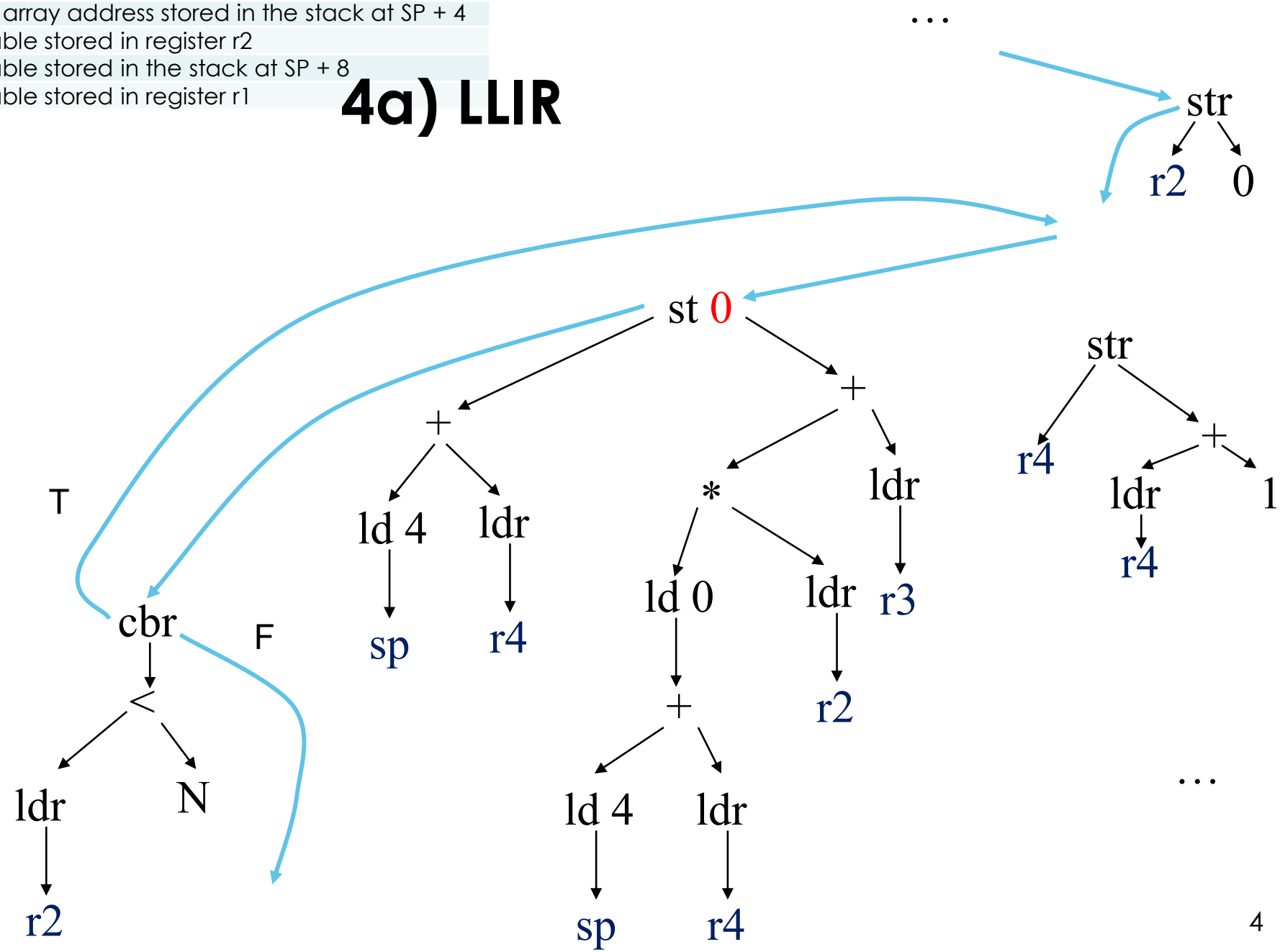
Group 4

- **4a) [2pt]** Indicate a low-level intermediate representation (LLIR) for the section of the code in the example on the left (where N represents a 32-bit *int* constant) based on expression trees, but considering the processor with instruction set presented below, and the type and storage of the variables given by the following table.

Var.	Type	Storage
A	int8 A[N] (array of bytes)	Base array address stored in the stack at SP + 4
i	int i (32-bit scalar var.)	Variable stored in register r2
m	int m (32-bit scalar var.)	Variable stored in the stack at SP + 8
x	int x (32-bit scalar var.)	Variable stored in register r1

4a) LLIR

```
i = 0;
m = A[0]
do {
    i=i+1;
    x = A[i];
    if (x > m)
        m = x;
} while(i<N);
```

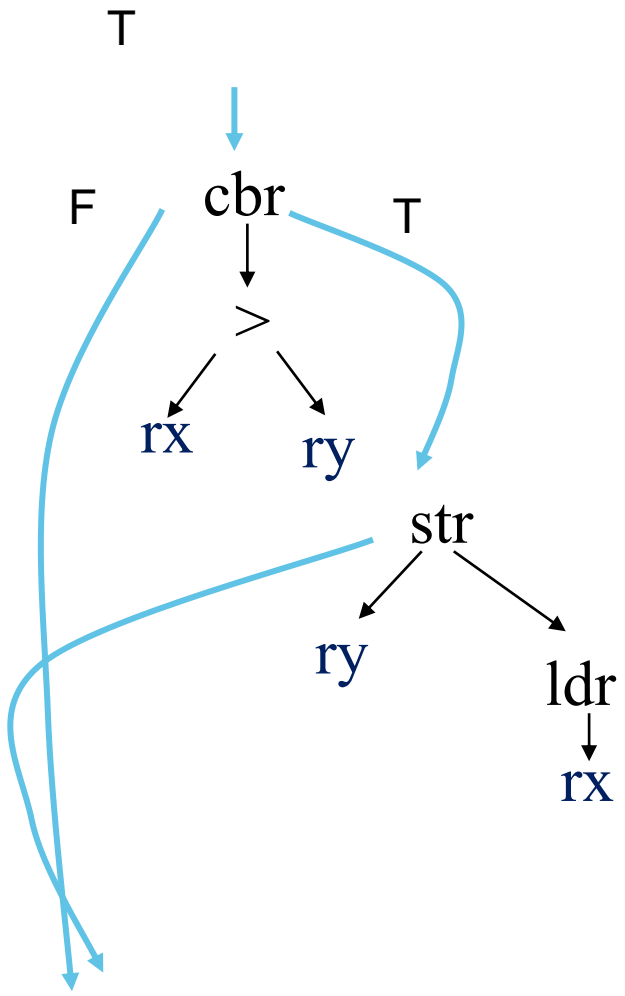


Notes: nodes "ldr" and "str" have been introduced to identify loads (reads) and stores (writes) from/to registers.

Group 4

- **4b) [1pt]** Considering that a new version of the processor includes an instruction able to implement the code in lines 5 and 6, show the pattern that can be used by the instruction selection algorithm (e.g., Maximal Munch) in order to have the option to associate that instruction to the CFG representation of those lines of code.

```
if (x > m)
    m = x;
```

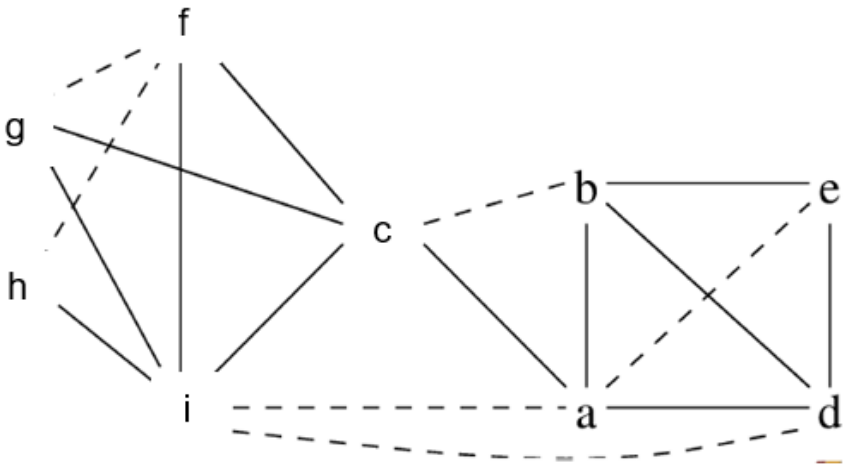


Group 4

- **4c) [2.5pt]** In the context of liveness analysis, and based on the dataflow analysis equations that are responsible to propagate information between nodes, present modifications to the dataflow analysis iterative algorithm (maintaining the input CFG) that may make it faster.

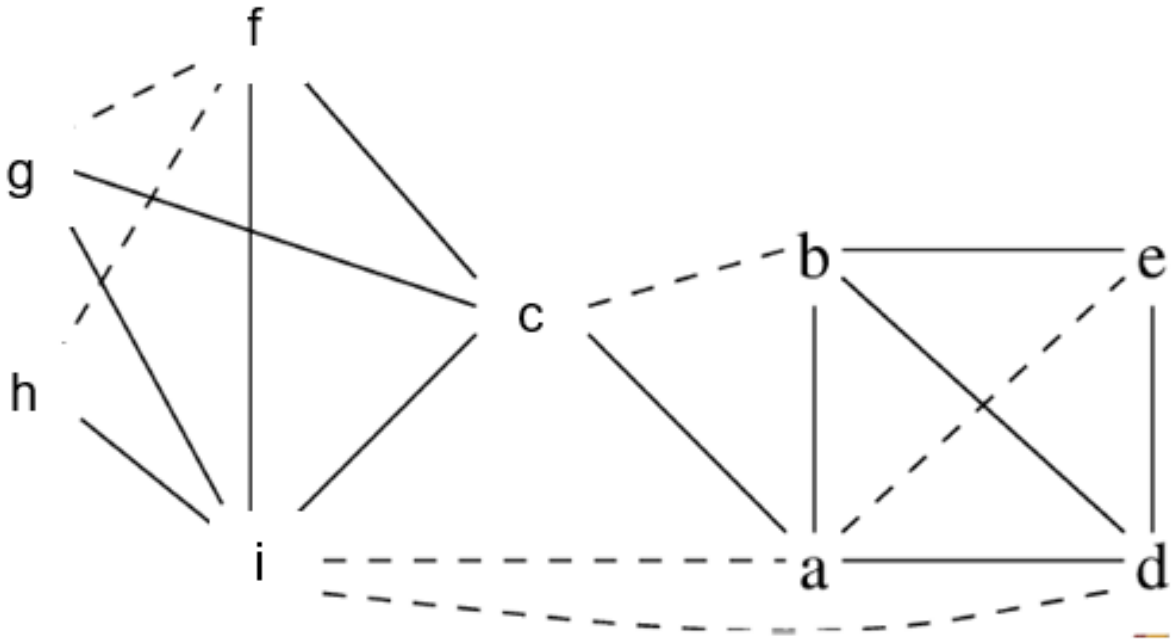
Group 4

- **4d) [2.5pt]** Consider the interference graph (IG) presented below. Indicate a possible allocation of registers using the graph coloring based register allocation algorithm presented in the lectures and considering a maximum of 3 registers (*R1, R2, and R3*). Show the steps performed and the content of the stack after the complete simplification of the IG. In the case of having to perform *spilling*, use the degree of the nodes to decide. In the case of having to perform *coalescing*, indicate the heuristic used and why you *coalesce* or *freeze*. Show the result of the first iteration of the register allocation algorithm (i.e., without repeating the process).



Group 4

- **4d)** a maximum of 3 registers (*R1, R2, and R3*). Show the steps performed and the content of the stack after the complete simplification of the IG.
- In the case of having to perform *spilling*, use the degree of the nodes to decide.
- In the case of having to perform *coalescing*, indicate the heuristic used and why you *coalesce* or *freeze*.



Group 5

- **5a) [2pt]** Considering the liveness analysis using dataflow analysis, members of a compiler team suggested the following two options: (a) merging CFG nodes that have a single predecessor and a single successor into a new node; (b) Instead of computing dataflow information for all variables at once using sets, compute the analysis for each variable separately. Comment on these two options and explain why you think each of the options might make sense and be advantageous or not.
- Both options are advantageous:
- Option a) allows to have CFG with less nodes and the need of a single iteration for liveness analysis in each of the new nodes.
- Option b) reduces the complexity to store the sets needed (use, def, in, out) and the operations needed in the dataflow analysis equations. However it needs to repeat the dataflow analysis algorithm per each scalar variable.

1b) Tree patterns for each machine instruction:

Instruction	Operation
load, Ri, Rj, C	Ri ← Mem[Rj+C]
store Rj, C, Ri	Mem[Rj+C] ← Ri
add Ri, Rj, Rk	Ri ← Rj + Rk
addi Ri, Rj, C	Ri ← Rj + C
mul Ri, Rj, Rk	Ri ← Rj * Rk
lth Ri, Rj, label1	If(Ri < Rj) goto label1
lte Ri, Rj, label1	If(Ri <= Rj) goto label1
gth Ri, Rj, label1	If(Ri > Rj) goto label1
gte Ri, Rj, label1	If(Ri >= Rj) goto label1
goto label1	jump to label1

...

1c): Selection of Instructions Using Maximal Munch

```
for(i=0; i < 100; i++)
    for(j=0; j < 200; j++)
        A[i][j] = 0;
```

1d) Considering you want to minimize execution time for Code1, which would be the template used for representing the FOR loops of the example?

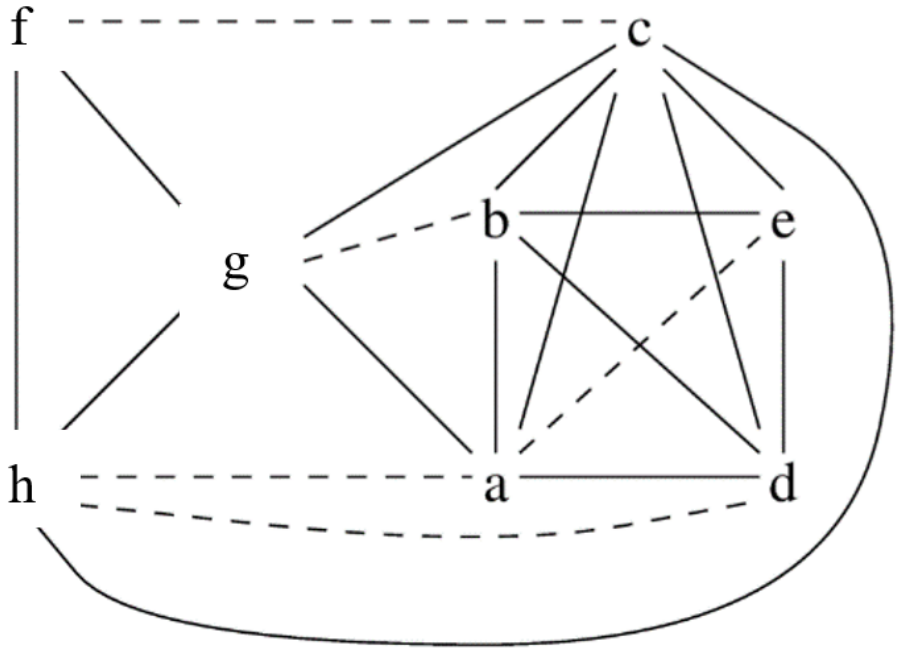
```
for(i=0; i < 100; i++)
    for(j=0; j < 200; j++)
        A[i][j] = 0;
```

```
LOOP1:

LOOP2
```

```
i=0;

j=0;


// code for loop body
 A[i][j] = 0;

j++;
if j < 200 goto LOOP2

i++
if i < 100 goto LOOP1
```

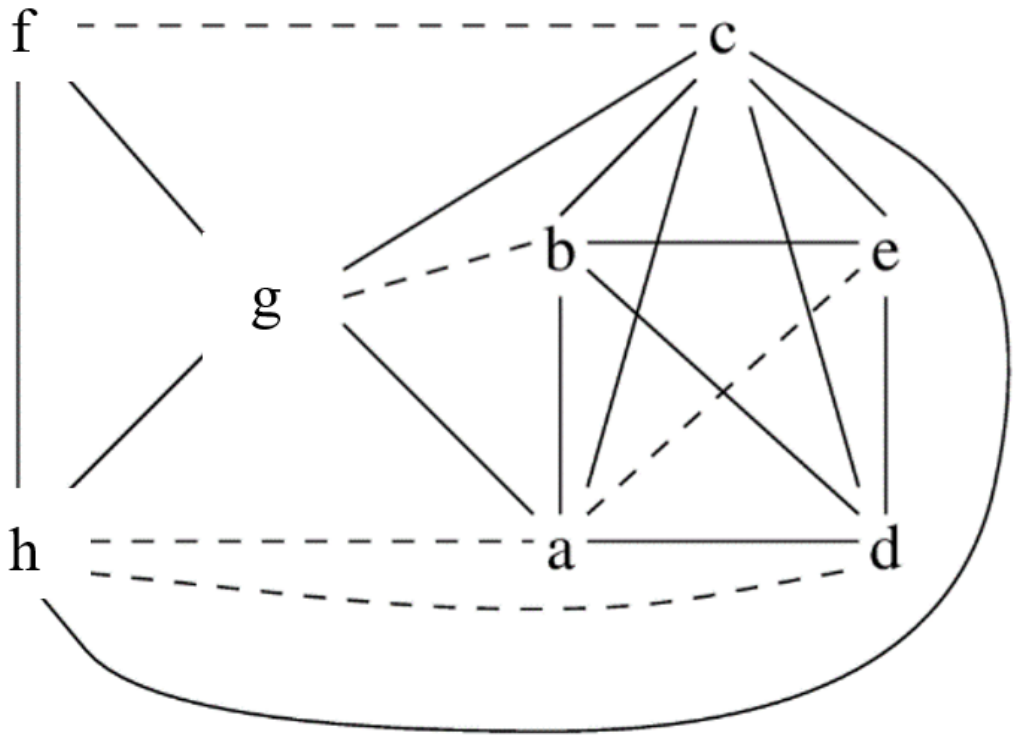
Group II: Scheduling and Register Allocation (9 pts)

- **2a)** [3pt] Consider the interference graph (IG) presented below. Indicate a possible allocation of registers using the graph coloring based register allocation algorithm presented in the course lectures and considering a maximum of 3 registers (*R1, R2, and R3*). Show the content of the stack immediately after the simplification of the IG. In the case of having to perform *spilling*, use the degree of the nodes to decide. In the case of register coalescing, justify the decision. Show the result of the first iteration of the register allocation algorithm (i.e., without repeating the process).



2a) Register Allocation

- R1, R2, R3 (K=3)



aeh
gb
f
d
Coallesce ae-h
Coallesce g-b
coallesce a-e
may spill c

After the first iteration of register allocation:

- Spill c
- R1 = {aeh, }
- R2 = {gb}
- R3 = {f,d}

2b) Criterion that you would suggest to integrate in a compiler for the selection of variables for spilling

- Important to include the frequency of the execution of the instructions, the number of def and uses
- Possible equation used per variable and the priorities given by ascending sort of the values:

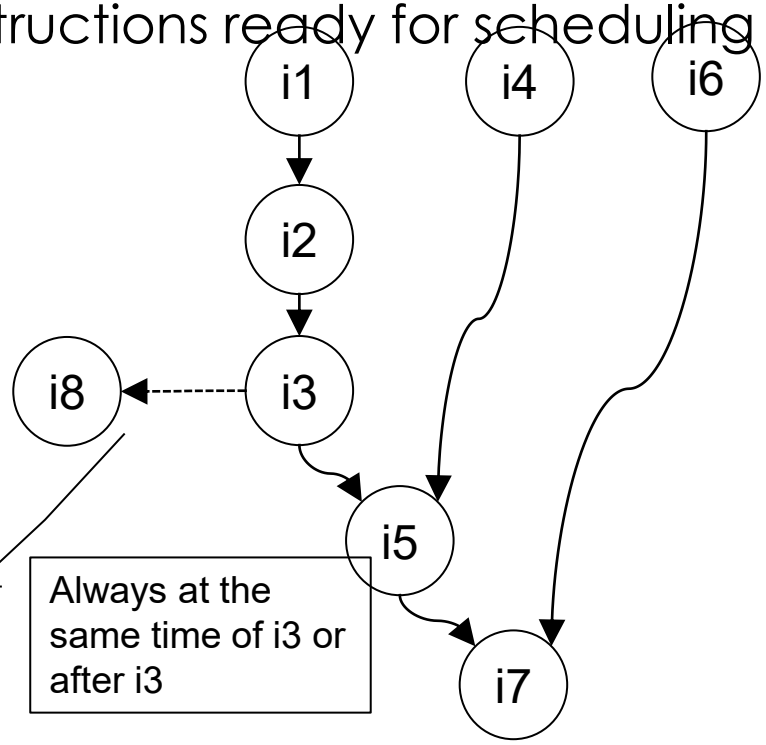
$$(\underbrace{\#uses + \#defs}_{\text{Outside loops}} + \underbrace{freq(\#uses) + freq(\#defs)}_{\text{According to the number of loop iterations}})/degree$$

2c) List Scheduling

- Considering that in the target machine all instructions execute in 1 clock cycle and the machine has one unit for load/store instructions and two units for the other instructions, present the scheduling of instructions resultant of applying the list-scheduling algorithm to the example in Code2. Present the data-dependence graph and the criterion to order the instructions ready for scheduling

Code2:

i1: addi r4, r0, 20
i2: mul r5, r4, r3
i3: add r6, r5, r2
i4: addi r7, r0, 4
i5: mul, r8, r7, r6
i6: load r9, SP, 4
i7: add r4, r8, r9
i8: addi r2, r2, 1



#cc	Load/store	FU1	FU2
1	i6	i1	i4
2		i2	
3		i3	i8
4		i5	
5		i7	

Group III
T/F and comments

- **3a) There is only a valid order for executing the register allocation and scheduling stages.**

False

- **3b) When doing dataflow analysis for liveness analysis, the order the CFG nodes are visited never influences the final results obtained and the interference graph built from the In/Out sets.**

True