Advanced Pipeline Techniques

Task 1. (register renaming)

```
i1:  D1 ← MEM [R0+0]
i2:  D1 ← D0 * D1
i3:  D4 ← D2 / D1
i4:  D2 ← MEM [R1+0]
i5:  D3 ← D0 + D2
i6:  D4 ← D4 - D3
i7:  MEM [R1] ← D4
```

(a) Draw the precedence graph of the instructions. Add a label to each arc according to the type of the dependency (RAW, WAR or WAW).

(b) Eliminate the WAW and WAR data dependencies with register renaming! Apply register renaming on each instruction systematically. Denote the *physical* floating point registers by `T0, T1, T2, …`, and the *physical* integer registers by `U0, U1, U2, …`. The initial content of the register alias table is the following:

Logical	Physical register
R0:	U7
R1:	U3
D0:	T6
D1:	T2
D2:	T8
D3:	T1
D4:	T11

If a new physical register is required, choose the one right after the biggest appearing in the register alias table.

(c) Draw the precedence graph of the instructions after register renaming.

Task 2. (register renaming)

```
i1:  D1 ← D2 - D3
i2:  D4 ← D1 + D2
i3:  D5 ← D4 + D1
i4:  D0 ← D2 / D5
i5:  D1 ← MEM [R0+0]
i6:  D4 ← D1 + D2
i7:  D5 ← D4 + D1
```

(a) Draw the precedence graph of the instructions. Add a label to each arc according to the type of the dependency (RAW, WAR or WAW).

(b) Eliminate the WAW and WAR data dependencies with register renaming! Apply register renaming on each instruction systematically. Denote the *physical* floating point registers by `T0, T1, T2, …`, and the *physical* integer registers by `U0, U1, U2, …`. The initial content of the register alias table is the following:

Logical	Physical register
R0:	U4
D0:	T6
D1:	T2
D2:	T8
D3:	T1
D4:	T9
D5:	T10

If a new physical register is required, choose the one right after the biggest appearing in the register alias table.

(c) Draw the precedence graph of the instructions after register renaming.

Task 3. (VLIW scheduling)

```
i1:  R2 ← MEM [R0+0]
i2:  R3 ← R0 * R2
i3:  R8 ← R4 / R3
i4:  R5 ← MEM [R1+8]
i5:  R6 ← R2 + R5
i6:  R9 ← R5 / R6
i7:  R10 ← R6 * R9
```

Assume we need to compile the program above on a VLIW processor where the following type of instructions can be placed in an instruction group:

- 2 memory load/store operations (latency: 3 cycles, iteration interval: 1 cycle)
- 2 integer arithmetic instructions (latency: 1 cycle)

(a) Draw the precedence graph of the instructions. Add a label to each arc according to the type of the dependency (RAW, WAR or WAW).

(b) Schedule the instructions for the given VLIW processor. Determine the content of the instruction groups and the time when the instruction groups can be executed.

(c) How much faster is the program compared to the plain 1-way case without pipeline?

(d) How many instruction groups are created for a classical and how many for a dynamic VLIW CPU?

Task 4. (VLIW scheduling)

```
i1:  R1 ← R2 - R3
i2:  R4 ← MEM [R1]
i3:  R6 ← R4 + R1
i4:  R0 ← R2 / R6
i5:  R7 ← MEM [R9]
i6:  R8 ← R7 + R2
i7:  R7 ← R7 + R6
```

Assume we need to compile the program above on a VLIW processor where the following type of instructions can be placed in an instruction group:

- 2 memory load/store operations (latency: 3 cycles, iteration interval: 1 cycle)
- 2 integer arithmetic instructions (latency: 1 cycle)

(a) Draw the precedence graph of the instructions. Add a label to each arc according to the type of the dependency (RAW, WAR or WAW).

(b) Schedule the instructions for the given VLIW processor. Determine the content of the instruction groups and the time when the instruction groups can be executed.

(c) How much faster is the program compared to the plain 1-way case without pipeline?

(d) How many instruction groups are created for a classical and how many for a dynamic VLIW CPU?