$$T_{CPU} = n_{clock\ cycles} * T_{clock} = \frac{IC*CPI}{f_{clock}} \text{ \# cpu time}$$

$$CPI = \frac{n_{clock\ cycles}}{IC} = \sum_{i=1}^{N_{classes}} CPI_i * \frac{IC_i \text{ \# relative probability}}{IC}$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{1-Fraction_{enhanced}+\frac{Fraction_{enhanced}}{Speedup_{enhanced}} \text{ \# this fraction is 0 to get speedup max}}$$

$$= \frac{1}{1-\alpha_{parallel}+\frac{\alpha_{parallel}}{N} \text{ \# this fraction is 0 for } \infty\text{-core cpu \# can have multiple fractions}}$$

$$Thoughput = Performance = \frac{Workload}{Execution\ Time} = \frac{W}{T_{exec}}$$

$$S_{x/y} = \frac{P_x}{P_y} = \frac{W_x}{W_y} = \frac{T_x}{T_y}$$

Deep-Parallelism (Pipelining, Super-Pipelining), Wide-Parallelism (more than one instruction or stream)

Non-pipelined code is a sum of all components. CPI=1. Pipelined code is the slowest stage. CPI>1

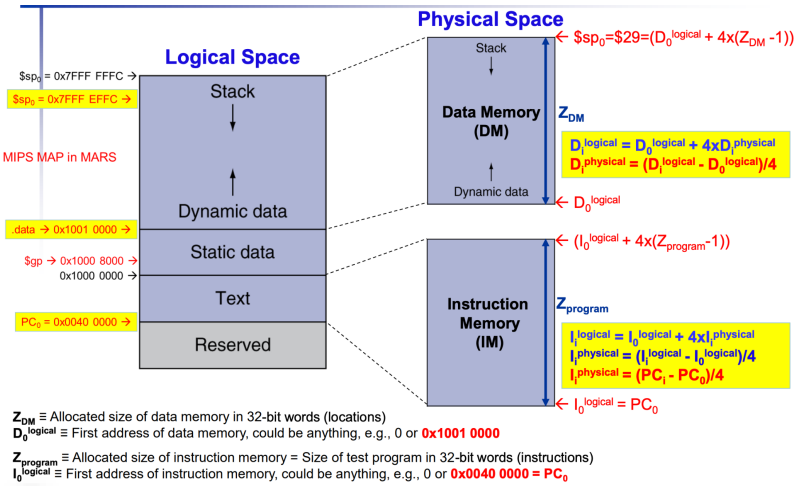
$$\Rightarrow T_{clock} = \left(\tau_{PC\_setup} + \tau_{PC\_clk-q} + \tau_{CU} + 3\tau_{mux}\right) + \left(\tau_{IM_{read}} + \tau_{RF_{read}} + \tau_{ALU} + \tau_{DM_{read}} + \tau_{RF_{write}}\right)$$

$$when \quad \tau_{IM_{read}} = \tau_{DM_{read}} = \tau_{mem} \quad , \quad and \quad \tau_{RF_{read}} = \tau_{RF_{write}} = \tau_{RF}$$

$$\Rightarrow T_{clock} = \left(\tau_{PC\_setup} + \tau_{PC\_clk-q} + \tau_{CU} + 3\tau_{mux}\right) + \left(2\tau_{mem} + 2\tau_{RF} + \tau_{ALU}\right)$$

Typically, limiting paths are: Memory, Register File, and ALU

$$\Rightarrow T_{clock} \cong 2\tau_{mem} + 2\tau_{RF} + \tau_{ALU}$$




$$t_r = t_{clk-q} + t_{setup} \text{???}$$

$$t_{exec}^{non-pipelined} = n_{cycles}^{non-pipelined} * T_{clock}^{non-pipelined} = N * \sum_{i=1}^{L} t_i = NLt_{avg}$$

$$P^{non-pipelined} = f = \frac{1}{T_{clock}^{non-pipelined}}$$

$$CPI^{pipelined} = \frac{n_{cycles}^{pipelined}}{N=IC} = \delta + \frac{L-\delta}{N} > 1;\ min = \delta.\ max = L$$

$$t_{exec}^{pipelined} = n_{cycles}^{pipelined} * T_{clock}^{pipelined} = (N + L - 1)(t_{max} + t_r) = (L + (N-1)\delta)(t_{max} + t_r)$$

$$P_{pipelined} = \frac{f_{clock}^{pipelined}}{CPI^{pipelined}} = \frac{1}{(\delta+\frac{L-\delta}{N})(t_{max}-t_r)};\quad if\ max,\ N = \infty$$

$$S = \frac{p^{pipelined}}{p^{non-pipelined}} = \frac{CPI^{non-pipelined}}{CPI^{pipelined}} \frac{f_{clock}^{pipelined}}{f_{clock}^{non-pipelined}} = \frac{1}{\delta+\frac{L-\delta}{N}} \frac{f_{clock}^{pipelined}}{f_{clock}^{non-pipelined}} = \frac{L}{\delta+\frac{L-\delta}{N}} \frac{t_{avg}}{t_{max}+t_r};\quad if\ max,\ N = \infty$$

$$t_i = t_{avg} = t_{max} = t \text{ \# balanced}$$

Hazards (structure - write same time, data - need result, control - branch)

$$n_{cycles}^{pipelined} = L + (N-1)\delta$$

$$1 \le \delta = 1 + \sum_{i=1}^{K_{hazard}} p_i^{hazard} p_i^{mispredict} \beta_i \le L \text{ where } \beta_i often = L_i - 1$$

β for branch mispredict with forwarding is 2

$$N_{min} = \frac{L-\delta}{fp/fn-\delta} \#min\ program\ size\ for\ pipelined\ to\ be\ faster$$

$$Faster\ if\ fp/fn > \delta\ (if\ long)\ or\ fp/fn > L\ (for\ any)$$

**Relations:**

$$C_{pipelined} = C_{non-pipelined} + Lc_r$$

$$T_{pipelined} = \frac{t_{max}}{t_{avg}} \frac{T^{non-pipelined}}{L} + t_r$$

$$P_{pipelined} = \frac{f_{clock}^{pipelined}}{CPI^{pipelined}} = \frac{1}{\delta+\frac{L-\delta}{N}} \frac{1}{T^{pipelined}}$$

$$PCR = \frac{P_{max}^{pipelined}}{C^{pipelined}} \#performance/cost\ ratio$$

$$= \frac{1}{\delta(C^{non-pipelined}+L_{optimal}c_r)(\frac{t_{max}}{t_{avg}} \frac{T^{non-pipelined}}{L_{optimal}}+t_r)}$$

SPECINT2000 benchmark:

- 25% loads
- 10% stores
- 11% branches
- 2% jumps
- 52% R-type

Suppose:

- 40% of loads used by next instruction
- 25% of branches mispredicted
- All jumps flush next instruction

**What is the average $CPI = CPI_{avg}$?**

- Load/Branch $CPI = 1$ when no stalling, 2 when stalling
- $CPI_{load} = 1(0.6) + 2(0.4) = 1.4$
- $CPI_{branch} = 1(0.75) + 2(0.25) = 1.25$

$$\boldsymbol{CPI_{avg}} = (0.25)(1.4) + (0.1)(1) + (0.11)(1.25) + (0.02)(2) + (0.52)(1)$$

$$= \mathbf{1.15}$$

$$\frac{\delta PCR}{\delta L} = 0 \Rightarrow L_{optimal} = \sqrt{\frac{C^{non-pipelined}}{c_r}\frac{T^{non-pipelined}}{t_r}\frac{t_{max}}{t_{avg}}} \text{ (floor)}$$

1045ps=956mhz

$lui\ \$s0,\ left16;\ ori\ \$s0,\ \$s0,\ right16$

Addressing modes: immediate, register, baseAndOffset, pcRelative, pseudoDirect

```
caller:
addi $a0, $zero, 18
lui $at, 4097
ori $a1, $at, 160
#$a1, $zero, 0x100100A0
jal fib_seq
next: j exit

fib_seq:
addi $sp, $sp, -16
sw $ra, 12($sp)
sw $t0, 8($sp)
sw $t1, 4($sp)
sw $a0, 0($sp)
addi $t0, $a0, 1
add $a0, $zero, $zero
loop:
beq $a0, $t0, break
sll $t1, $a0, 2
add $t1, $a1, $t1
jal fib
sw $v0, 0($t1)
addi $a0, $a0, 1
j loop
break:
lw $a0, 0($sp)
lw $t1, 4($sp)
lw $t0, 8($sp)
lw $ra, 12($sp)
addi $sp, $sp, 16
jr $ra

fib: addi $sp, $sp, -12
sw $ra, 8($sp)
sw $a0, 4($sp)
bne $a0, $zero, L1
add $v0, $zero, $zero
addi $sp, $sp, 12
jr $ra
L1: addi $v0, $zero, 1
bne $a0, $v0, L2
addi $sp, $sp, 12
jr $ra
L2: addi $a0, $a0, -1
jal fib
sw $t0, 0($sp)
add $t0, $zero, $v0
addi $a0, $a0, -1
jal fib
add $v0, $t0, $v0
lw $t0, 0($sp)
```

```
lw $a0, 4($sp)
lw $ra, 8($sp)
addi $sp, $sp, 12
jr $ra

fib_seq:
sw $zero, 0($a1)
bne $a0, $zero, nonZero
add $v0, $zero, $zero
jr $ra
nonZero:
addi $v0, $zero, 1
sw $v0, 4($a1)
bne $a0, $v0, prepareLoop
j break
prepareLoop:
addi $sp, $sp, -16
sw $t0, 12($sp)
sw $t1, 8($sp)
sw $t2, 4($sp)
sw $a0, 0($sp)
addi $t0, $a0, 1
addi $a0, $zero, 2
loop:
beq $a0, $t0, break
sll $t1, $a0, 2
add $t1, $a1, $t1
lw $t2, -4($t1)
lw $v0, -8($t1)
add $v0, $t2, $v0
sw $v0, 0($t1)
addi $a0, $a0, 1
j loop
break:
lw $a0, 0($sp)
lw $t2, 4($sp)
lw $t1, 8($sp)
lw $t0, 12($sp)
addi $sp, $sp, 16
jr $ra
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE work.my_package.ALL;

ENTITY mips_single_cycle IS
   PORT(
      clk : IN     std_logic;
      rst : IN     std_logic
   );
END mips_single_cycle ;

ARCHITECTURE struct OF mips_single_cycle IS
   SIGNAL PC_next  : std_logic_vector (n_bits_address - 1 DOWNTO 0);
   SIGNAL CU_ALUSrc     : std_logic;
  COMPONENT PC_register
    PORT(
       PC_next    : IN  std_logic_vector (n_bits_address - 1 DOWNTO 0);
       clk        : IN  std_logic;
       rst        : IN  std_logic;
       PC_current : OUT std_logic_vector (n_bits_address - 1 DOWNTO 0)
    );
  END COMPONENT;
BEGIN
   PC_register_inst : PC_register
   PORT MAP (
     PC_next    => PC_next,
     clk        => clk,
     rst        => rst,
     PC_current => PC_current
   );
   PC_inc <= std_logic_vector(SIGNED(PC_current) + 4);
   pseudo_address <=
        InstrMem_Instr(pseudo_address_end DOWNTO pseudo_address_start);
   branch_taken <= CU_BEQ AND ALU_zero;

   PC_cond_branch_proc : PROCESS (
                    branch_taken, PC_inc, immediate_Sign_Extended)
   BEGIN
     IF branch_taken = '1' THEN
       PC_cond_branch <= std_logic_vector(SIGNED(PC_inc) +
                SIGNED(shift_left(SIGNED(immediate_Sign_Extended), 2)));
     ELSE
       PC_cond_branch <= PC_inc;
     END IF;
     CASE opcode IS
       WHEN 0 => END CASE;
       WHEN others => END CASE;
     END CASE;
   END PROCESS PC_cond_branch_proc;
END struct;
```

( (ID/EX.MemRead = '1') and
( (ID/EX.RegisterRt = IF/ID.RegisterRs) or
(ID/EX.RegisterRt = IF/ID.RegisterRt) ) )

detected

stall pipeline for 1-cycle (insert a bubble)

MEM hazard

- if ( (MEM/WB.RegWrite = '1') and (MEM/WB.RegisterRd ≠ 0) and
  NOT ( (EX/MEM.RegWrite = '1') and (EX/MEM.RegisterRd ≠ 0)
  and (EX/MEM.RegisterRd = ID/EX.RegisterRs) )
  and (MEM/WB.RegisterRd = ID/EX.RegisterRs) )
  then ForwardA = "01"; end if;
- if ( (MEM/WB.RegWrite = '1') and (MEM/WB.RegisterRd ≠ 0) and
  NOT ( (EX/MEM.RegWrite = '1') and (EX/MEM.RegisterRd ≠ 0)
  and (EX/MEM.RegisterRd = ID/EX.RegisterRt) )
  and (MEM/WB.RegisterRd = ID/EX.RegisterRt) )

EX hazard → forward Rd from EX/MEM fence

- if ( (EX/MEM.RegWrite = '1') and (EX/MEM.RegisterRd ≠ 0)
  and (EX/MEM.RegisterRd = ID/EX.RegisterRs) )
  then ForwardA = "10"; end if;
- if ( (EX/MEM.RegWrite = '1') and (EX/MEM.RegisterRd ≠ 0)
  and (EX/MEM.RegisterRd = ID/EX.RegisterRt) )
  then ForwardB = "10"; end if;

```
fact:   addi   $sp, $sp, -12        #adjust stack pointer
        sw     $ra, 8($sp)          #save return address
        sw     $a0, 4($sp)          #save argument n
        sw     $t0, 0($sp)          #save $t0
        slti   $t0, $a0, 1          #test for n < 1
        beq    $t0, $zero, L1       #if n >=1, go to L1
        addi   $v0, $zero, 1        #else return 1 in $v0
        lw     $t0, 0($sp)          #restore $t0
        addi   $sp, $sp, 12         #adjust stack pointer
        jr     $ra                  #return to caller (1st)

L1:     addi   $a0, $a0, -1         #n >=1, so decrement n
        jal    fact                 #call fact with (n-1)
        #this is where fact returns
bk_f:   lw     $t0, 0($sp)          #restore $t0
        lw     $a0, 4($sp)          #restore argument n
        lw     $ra, 8($sp)          #restore return address
        mul    $v0, $a0, $v0        #$v0 = n * fact(n-1)
        addi   $sp, $sp, 12         #adjust stack pointer
        jr     $ra                  #return to caller (2nd)
```

*Recurrence equation (Non – Homogeneous):*

$y(n) = y(n-1) + 14 \rightarrow$ *equation order* $= N = 1$

$y(n) - y(n-1) = 14,\ y(0) = 10$

*Homogeneous recurrence equation:*

$y(n) - 2y(n-1) + y(n-2) = 0 \rightarrow$ *equation order* $= N = 2$

$y(n) - 2y(n-1) + y(n-2) = 0,\ y(0) = 10,\ y(1) = 24$

*Characteristic equation:*

$\lambda^n - 2\lambda^{n-1} + \lambda^{n-2} = 0 \rightarrow \lambda^{n-2}\left(\lambda^2 - 2\lambda + 1\right) = \lambda^{n-2}(\lambda - 1)^2 = 0$

$\rightarrow$ *number of roots* $= N_{roots} = 1,\ \lambda_1 = 1$ *with root multiplicity* $m_1 = 2$

*General form of solution:* $y(n) = \sum_{i=1}^{N_{roots}} \left( \left( \sum_{j=0}^{m_i - 1} c_{i,j} n^j \right) \lambda_i^n \right)$,

*where* $m_i$ = *multiplicity of root* $i$

$\rightarrow y(n) = \sum_{i=1}^{1} \left( \left( \sum_{j=0}^{2-1} c_{i,j} n^j \right) \lambda_i^n \right) = \left( \sum_{j=0}^{1} c_{i,j} n^j \right) \lambda_1^n = \left( c_{1,0} + c_{1,1} n \right) \lambda_1^n = c_0 + c_1 n \equiv O(n)$

*applying the initial conditions, i.e.,* $(y(0) = 10,\ y(1) = 24)$,

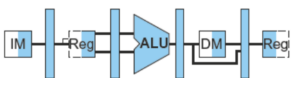
*and solving for* $c_0$, *and* $c_1 \Rightarrow y(n) = 10 + 14n \equiv O(n)$

*Total Execution Time:*

$t_{total} = IC \cdot CPI_{avg} \cdot T_{clock},\ IC = y(n)$

$t_{total} = y(n) \cdot CPI_{avg} \cdot T_{clock}$

$\Rightarrow t_{total} = (10 + 14n) \cdot CPI_{avg} \cdot T_{clock}$




```
LW   R1,0(R1)      | WB
LW   R1,0(R1)      | EX  MEM WB
BEQ  R1,R0,Loop    | ID  *** EX  MEM WB
LW   R1,0(R1)      | IF  *** ID  EX  MEM WB
AND  R1,R1,R2      |         IF  ID  *** EX  MEM WB
LW   R1,0(R1)      |             IF  ID  *** ID  EX  MEM
LW   R1,0(R1)      |                 IF  *** ID  ID  ***
BEQ  R1,R0,Loop    |                     IF  *** IF  ***
```

| EX to 1st Only | MEM to 1st Only | EX to 2nd Only | MEM to 2nd Only | EX to 1st and MEM to 2nd | Other RAW Dependences |
|---|---|---|---|---|---|
| 5% | 20% | 5% | 10% | 10% | 10% |

```
LW   R2,0(R1)      | IF  ID  EX MEM WB
BEQ  R2,R0,Label2 (NT)     IF  ID *** EX  MEM WB
LW   R3,0(R2)                  IF  ID  EX  MEM WB
BEQ  R3,R0,Label1 (T)              IF  ID  *** EX  MEM WB
BEQ  R2,R0,Label2 (T)                  IF  *** ID  EX  MEM WB
SW   R1,0(R2)                              IF  ID  EX  MEM WB
```

If Branch: 2nd load or 1st ALU=1; 1st load=1