Computer Architecture 1

Fanglin Xu

Learning Log

2023.04.12

1. Addressing Modes. It contains absolute, indirect and base, of which I think the most difficult to understand is indirect.
   - Absolute. It uese the immediate value as address. Such as `mov eax, (1000)`, which means first I have to find what is in Mem[1000] and second passes it to eax.
   - Indirect is not the same it uses the intermediate memory access(存储器) such as registers, which is further classified into two catagories: **memory indirect and register indirect**.
   - Memory indirect. `mov eax, ((r))`, uses Mem[reg[r]] as address, `reg[r]` means to get what is in register r. Finally, the whole thing is Mem[Mem[reg[r]]] to eax. Accessing memory twice.
   - Register indirect. `mov eax, (eax)`, uses reg[eax] as address, then it will give Mem[reg[rax])] to eax. Accessing memory only once.
   - So remember, every address mode finally is heading to memory. Beacuse the data is in memory.
2. Pointer. There is the most important, I should review it and find more tests.

When I was doing 18-447's homework, I encountered this:

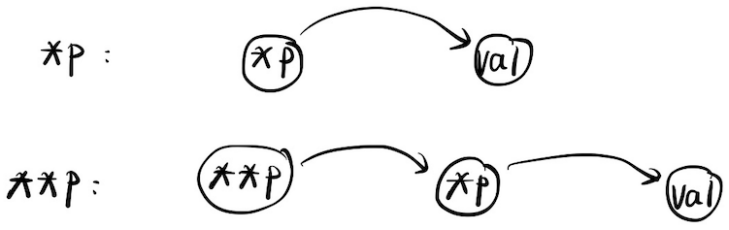
- ```
  int *p;  // *p is allocated in memory
  *p = 150;
  ```
- ```
  int **p; // *p and **p are allocated in memory
  **p = 150;
  ```

It asks me what kind of addrssing mode will I choose to have the least instructions if the value of p is in a register.

Solution:

The first one is register indirect and the second is memory indirect.

I think you should figure out what is `*p` and `**p`:

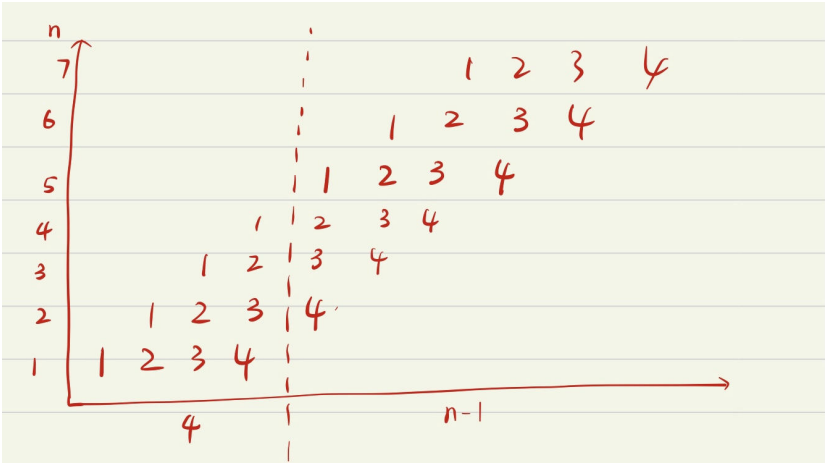


So the in `*p`, the value of p is the address of integer `val`, we just need one step which is `reg[p]`, then you use this to find the value in memory. But as to `**p`, the value of p is the address of the address of integer `val`, so we need `Mem[reg[p]]`, then you use this to find the value in memory, which needs 2 times access into memory.

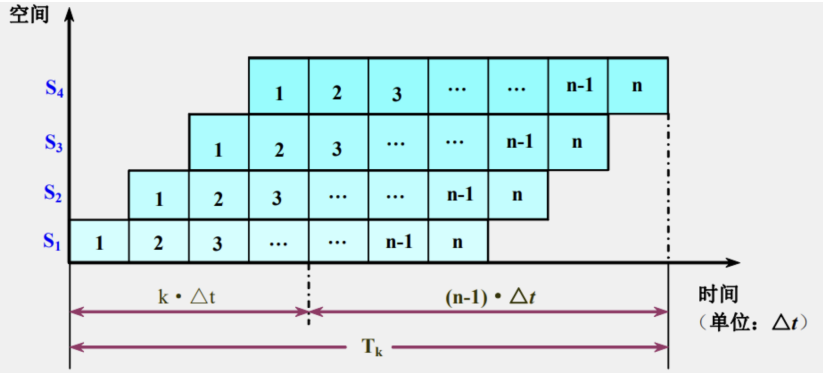
3. What is Register?. Register is in the CPU and is **directly connected to ALU**, so the latter one can use it for faster computation. It's faster but the number of it is limited.
4. Types of instruction. I should know that the instruction must have operand, otherwise CPU won't know what to do. Add? Sub? So add is 01 and sub is 00, they won't conflict.
   - Zero operand.
   - One operand.
   - Two operands.

2023.04.13

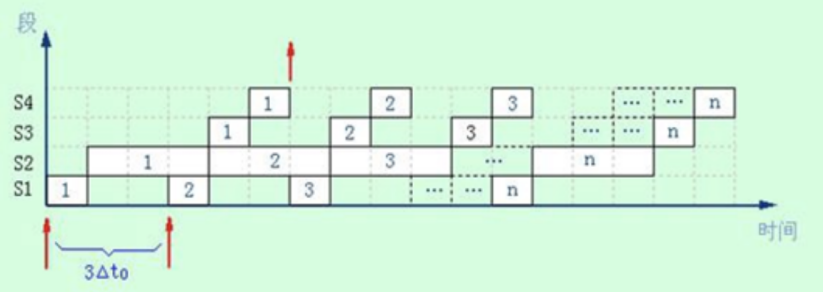
1. Speedup in pipeline. Assume that NP represents the time of non-pipeline while the P repersents the time of pipeline. So the speedup $\alpha = \frac{NP}{P}$
   - How to compute P and NP? Assume that we have n instructions, and the depth of pipeline is k, t_i represents the time i-th instruction needs.
   - NP NP is easy to compute, which is serial. So $NP = n \times \sum_{i=1}^{i=k} t_i$.
   - P P is a little tricky. I give the conclusion first, t_m represents the longest stage the pipeline has. **So $\boldsymbol{P = k \times t_m + (n-1) \times t_m = (n+k-1) \times t_m}$**. I make a image to show the situation when k = 4 below. The idea is to split it into two stages, the first is the whole instruction which has k steps and the second remains $n-1$ steps, you can take it as a rule, so just bear it in mind.



   - Finally, a pipeline would not have just 8 or 10 instructions, which means we should consider a scenario when $n \to \infty$. $\alpha = \frac{n \times \sum_{i=1}^{i=k} t_i}{(n+k-1) \times t_m} = \frac{\sum_{i=1}^{i=k} t_i}{\left(1+\frac{k-1}{n}\right) \times t_m}$. When $n \to \infty$, $\alpha \to \frac{\sum_{i=1}^{i=k} t_i}{t_m}$. An ideal pipeline should have the same time in each stage. So the ideal speedup is k,which is the depth of pipeline, due to $t_1 = t_2 = ... = t_k = t_m$. But in our real lift, due to a lot of reasons such as access memory is always slower than computation, our pipeline speedup would not meet k.
2. Throughput in pipeline
   - What is it? The number of tasks a compeleted by pipeline per unit time. So TP= $\frac{n}{t}$.
   - The same as speedup, if every stage has the same time, then $t = (k+n-1) \times \Delta t$. **So the ideal TP is $\frac{1}{\Delta t}$.**

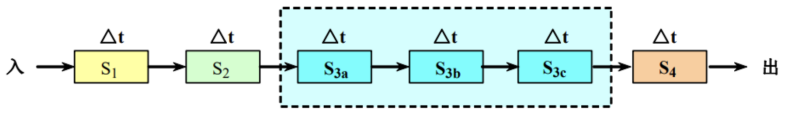


- Don't have same time? $t = \sum_{i=0}^{i=k} t_i + (n-1) \times t_m$. To get this result, you can move the last two n square to the first two 1 square, and then regard each big square of length 3 as a unit. **As a result, when $n \to \infty$, NP = $\frac{1}{t_m}$, what we call is the bottleneck**. You can see the bottleneck will affect pipeline throughput.

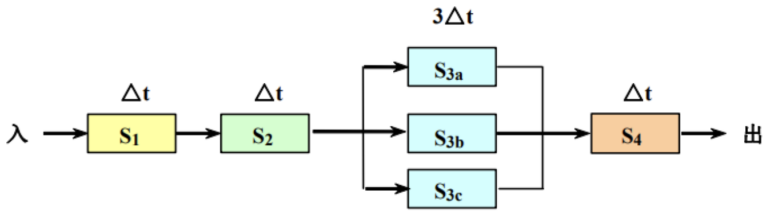


3. <u>How to mitigate(缓解) bottleneck in pipeline?</u>
   - What is bottleneck? The ideal pipeline always needs the same time in each stage, but in our instruction, Mem is slower than compute. So the Mem is the bottleneck in this pipeline. **The longest stage in pipeline is the bottleneck.**
   - Two ways to mitigate.

     1.Continue pipeline. For example, you have to open the refrigerator, put the elephant into it and close it. So why not continue spliting it into 3 steps?

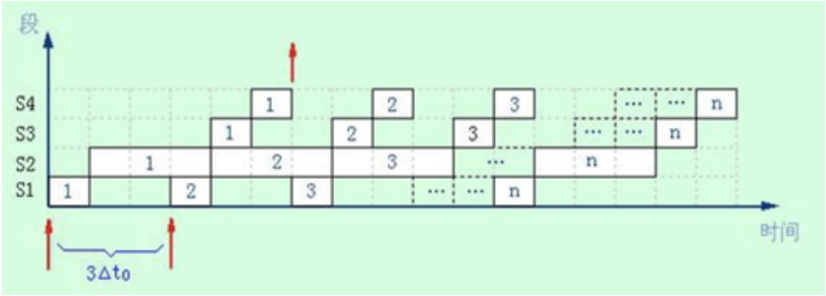


     2.Add more hardware. What if the stage can't be split? It means it takes me 30 minutes to cook the chicken, I cannot stop if I just cook a half. In this scenario, add more chefs(hardware) is a best way.



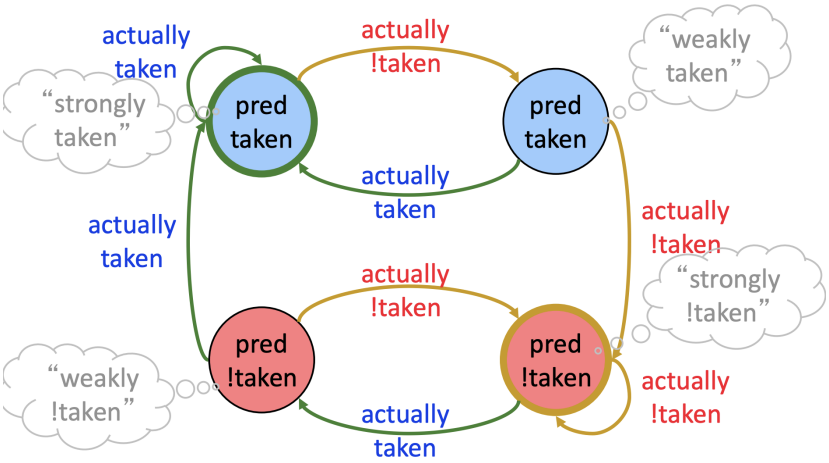
4. <u>Efficiency of pipeline</u>

- What is Efficiency? The ratio of actual time of using hardware to total running time. Or usage in other words.
- Same time every stage. $e_1 = e_2 = ... = e_n = \frac{n\Delta t}{(n+k-1)\Delta t} = \frac{n}{n+k-1}$. Get 1 when $n \to \infty$.
- Not the same time. the ratio of the square of white to total square.实际的面积除以总体的面积。



- $E = \frac{S}{k}$, S =Tp $\times$ Δt.

5. Two bits Predication.
   - T→NT is too fast, and if we encount a sequence like TNTNTNTN, the accuracy is 0%.
   - The reason of it is when facing a different we change our state too quicky. We could solve this by having two bits. Then we will have 50% in TNTNT..



And we would have 11 in strongly taken state, and switch to 10 if actually we don't take this and switch to 01 if actually we take this from state 00.