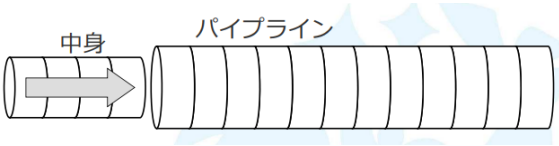
アーキテクチャ第９回予習資料　　第９回の授業までに、空欄を埋めて予習。　　質問は事前＋当日受付けます

第9回　パイプラインの考え方・パイプラインのハザード

９－１．　パイプラインの原理



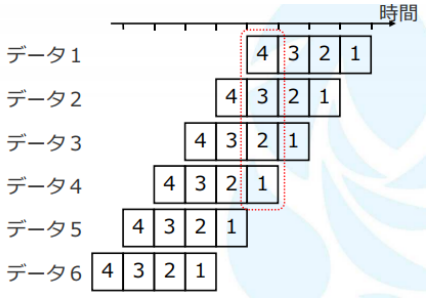
a）　パイプラインについて、空欄を埋めてください。

右図で、処理する機械は右側の「パイプライン」の部分で、その中は（　　）長さに分かれていて、それぞれを（　　）と呼ぶ。 他方、処理する対象となるデータは左側の「中身」と書かれたもので、これが（　　）の中を通過するうちに処理される。

パイプラインが S1, S2, S3, S4 の 4 段から成るとき、処理対象はパイプラインを通過しながら（　　）のように処理される。 処理対象が1つだけであれば、パイプラインのそれぞれの（　　）を順番に通過するだけであるが、図のように処理対象がデータ 1、データ 2、…、データ 6 のように複数あると、

パイプラインの S1 で見ていると、データは（　　）の順で通過し、S1 の処理を受ける。 S2…S4 も同様である。

他方、データの立場で見ると、データ 1 は（　　）の順で処理を受ける。 データ 2 は（　　）の順で処理を受けるが、そのタイミングは（うまく説明せよ　　）。その様子を描いたのが右の図（時間軸は普通とは左右が逆で、右ほど古い）である。



パイプラインのポイントは、

・ S1, S2, … が（　　）に対して自分の処理すること

・ 処理が済んだら一斉にデータが（　　へ）一段分（　　）こと

この「一斉に」が大事なポイントである。

９－２．　パイプラインの処理性能

パイプラインは、処理時間を短くします。（つまり、処理性能を向上させます。）

ステージ数を S、データの数を N としたときの、パイプラインによる性能向上を（自分の手で）計算してみましょう。

a.　パイプラインを使わないで、N 個のデータに対して、S 個の処理をするのにかかる時間はいくらですか？　但し、1 つのデータに 1 つの処理をする時間を T とします。

b.　パイプラインを使って、９－１の下側の図のように処理が進んでいるとします。 この時の 1 ステージ進む時間は

なので、データ 1 が S ステージのパイプを通過する時間は

になります。 では、この図にあるように、6 つのデータ（データ 1～データ 6）が通過する時間はどうなっているでしょうか

その計算方法は、

データ 6 がステージ 1 に入るまでの時間は、それ以前に順番に入るデータの個数が　　なので

その後、データ 6 がパイプラインを通過するので、それにかかる時間は

上の2つを足すと

では一般的にステージ数が S、データ数が N で、S<N の場合、2 つの合計は

c.　パイプラインによる速度向上の比率（つまり時間の短縮率、時間がどれだけ短縮されたか）は、

直列でかかる時間／パイプラインでかかる時間　＝

（注意） 欲しいのは、速度向上比＝（パイプラインでの速さ）／（直列での速さ） ですが、計算したのはそれぞれの「所要時間」であって、速さ＝1／時間なので、

速度向上比＝（1／（パイプラインでの時間））／（1／（直列での時間））

＝（直列での時間）／（パイプラインでの時間）

d.　パイプラインのスタート時（データがまだパイプ全体に埋まっていない期間）と、終了時（データがもう届かなくてパイプの一部しか使っていない期間）を取り除いて考えたいとしましょう。 それを計算する簡単な方法は、データが無限に続くと仮定することです。 つまりＮ → ∞ の時の速度向上比を計算してみましょう。

.....

.....

.....

結果は、直列に比べてＳ倍早くなることになります。

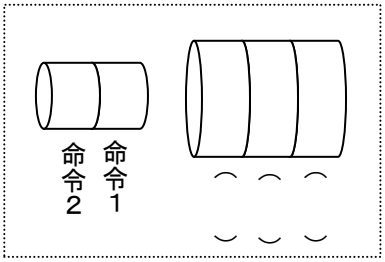
これは、次のように解釈できます。 パイプラインではＳ個の処理ユニットが同時に動作しているので、「S 並列」です。それなら（S 倍の人数をかけて処理をするのだから）処理能力は S 倍早くなるはずですよね。

９－３.　パイプラインの実例

1） 命令パイプライン

命令の実行サイクルは（　　）⇒（　　）⇒（　　）⇒（　　）⇒先頭へ戻る 。

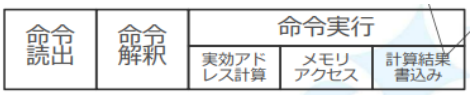
サイクルのそれぞれのステージは、互いに（　　）して動作できます （依存するところがない） 。



従って、右図のようにパイプライン動作をすることができます。 図中カッコ内にことば（ステージの内容）を入れてください。 但し実行サイクルの4番目はパイプラインのステージに作るまでの時間がかからないので書きません。

命令１は、パイプのステージ1から順に2，3と処理されます。これは命令の実行サイクルと同じです。 命令1に続いて2，3…と次々とパイプラインに入り処理されます。

９－２で計算したように、もし命令の数が無限に続けば、この例のパイプラインでは処理性能は（　　）倍になります。

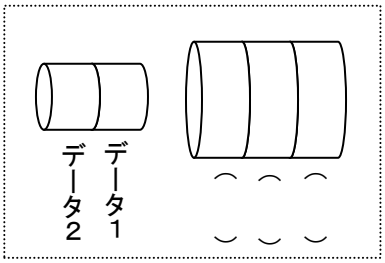


命令の実行サイクルがすべて1ステージの時間で終わるわけではなく、特に「実行」のステージは命令の内容によって余分に時間がかかります。この場合はパイプラインのステージを複数ステージに分けます。 右図は実行ステージを3つに分けた例で、全体で5ステージになります。こうするとパイプラインの処理性能は（　　）倍になります。 パイプラインのステージの長さ（時間）はすべて等しくないと、一斉にデータを1ステージ進めることができないので、等しくなければなりません。長い所（この例では実行ステージ）の長さ3Tに他のステージを合わせてすべて3Tにすると、3T×3ステージになり、一斉に進むことができますが、ステージ1と2では空き時間の無駄ができてしまいます。右図にあるように実行だけ3段にして全体を5段にすれば、空き時間の無駄はできないし、並列度もステージの数の分だけ増えます。

実際、PC で使われている Intel の x86 系プロセッサでは最近のもので 14～16 ステージ、一時は 30 ステージのものも作られました。

2） データパイプライン

ベクトルや行列などの要素データに対して同じ演算処理を行う場合に使われます。演算処理が処理 1→処理 2→処理 3 のように多段階になっているとき、ベクトルの要素データを次々に送り込むことによって、同じ処理をさせます。



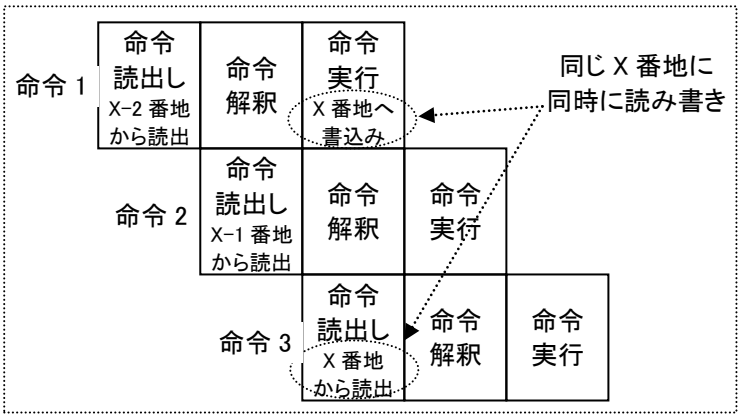
この場合、データの供給元(と行先)のために「ベクトルレジスタ」を設け、そこに処理対象のベクトルデータを入れておきます。また処理は(ハードウェアで出来ているために)組合せがあらかじめ限られるのが普通です。
このようなデータパイプラインは、PC に入っている Intel の x86 系プロセッサではマルチメディア用命令(音や画像データのそれぞれの要素に同じ命令処理を加える)で用意されているほか、グラフィックプロセッサ(GPU、すべての画素に対して同じ処理、すべての多面体に対して同じ処理などの場面で)に使われています。 昔(メインフレームの時代)にはベクトルのパイプライン演算機能を強化したスーパーコンピュータが使われました(Cray 社の Cray-I や II など)が、今は見かけません。代わりにグラフィックプロセッサ(GPU)を使って汎用のベクトル計算をする GPGPU の技術が広まっています。

9－4. パイプラインのハザート

a. パイプラインのハザードとは(　　)です。ハザードが起きるとパイプラインが(　　)してしまうので、期待した性能が出なくなります。
ハザードを分類すると、3種類あると言われます。区別はやや分かりにくいので、無理に覚えることは無いと思います。とにかく、どういう風になったときが問題なのか、説明できるようにしてください。 一応、3種類の名前を書いてみましょう。

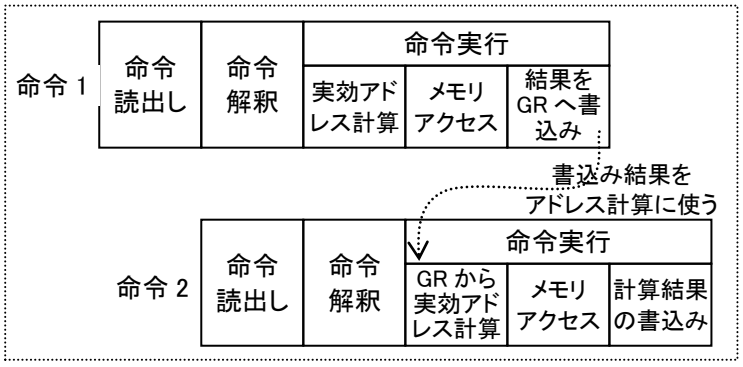
＊(　　)ハザードとは、(　　)とき
＊(　　)ハザードとは、(　　)とき
＊(　　)ハザードとは、(　　)とき

b. 構造ハザードの具体例
右の図は構造ハザードが起こっている例です。
何が問題なのか、説明してください。



解決策を、説明してください。

c. データハザードの例
右の図はデータハザードが起こっている例です。
何が問題なのか、説明してください。

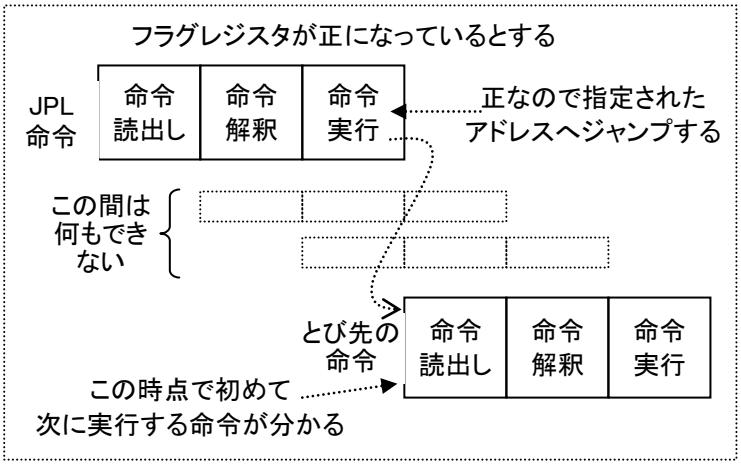


解決策を、説明してください。

d.　制御ハザードの例

右の図はデータハザードが起こっている例です。

何が問題なのか、説明してください。



解決策を、説明してください。

e. 制御ハザードの対策

制御ハザードの場合は、次の命令（とび先の命令）を待たせることによる遅れを救うための、いくつかの対策があります。

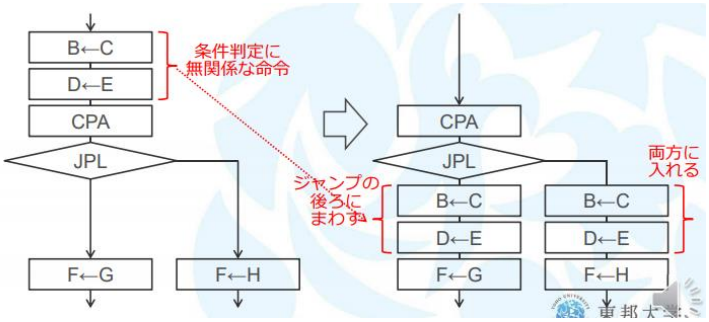
ここは、覚えなくても結構です。ふ～んこんな方法があるのか、と思っておいてください。

(1) 遅延分岐

上記の図の「この間は何もできない」という部分に、＜無関係な＞命令を入れる。

この作業は、機械語でプログラムを書いているときはプログラマが、また高級言語で書いているときにはコンパイラが判断して、行います。

これによって、タイムスロットは埋まるわけで、時間の無駄は起こらなくなります。問題は＜無関係な命令＞がどういうもので、それが好都合に存在するのか、ということです。



(2) 分岐予測

どちらに分岐するかを予測して、とび先が確定するより前に（上図の「この間は何もできない」の所で）予測したとび先にある命令を読み出し始めてしまう、というやり方です。 つまり、まだ判断の結果が分からないのに、たぶんこっちだという予測によって始めてしまいます。判断の結果が違っていれば予測で始めた読み出しや解釈はキャンセルします。キャンセルしたとしても、何もしないで判断結果を待つ場合と同じ遅れなので、損はしていません。

予測の方法として、ハードウェアでできるのは、その分岐命令でこの前に分岐したときの結果を覚えておくことぐらいです。（実際にそういう方法が使われています）