アーキテクチャ　（月曜クラス）　中間試験コメント　　2016-12-12

［1］ ア）イ）は2の補数を作る手続き　　第2回予習ビデオ 「マイナスの数の表現」 を参照

ウ）エ）は、左端ビットが1なので負の数（問題に「符号付き」） 10進への変換は第2回ビデオ「マイナスの数おまけ」を参照

オ）カ）キ）は、第3回ビデオ 「小数の表現－固定小数点」 を参照。 計算間違いが多かったので、問題だと思う。

なお、キ） $0.8_{10} = 0.1_{10} \times 8$ と考えることもできる。 だとすると、 0.1_{10} を左へ 3 ビットシフトしたものになる。

ク）は、見たことがない問題だと思う。 浮動小数点の仕組みをわかっているか見たかった。

仮数部が$0.111_2 = 0.875_{10}$、指数部が$10_2 = 2_{10}$ というので、表す数は、$0.875 \times 2^2 = 0.875 \times 4 = 3.5$

［2］ ア） プログラムをコンピュータ内（メモリ内）に保持すること。

イ） 「プログラムを内蔵しない方式の例を1つ挙げて」 という問題文にきちんと答えて欲しい。 出題の意図は、内蔵しない場合との比較を、具体的に説明してもらうことにある。 たとえば、内蔵しない例として「（単能機の）電卓」を挙げて、コンピュータだとプログラムを入れ替えて他の仕事ができるとか、電卓だと人間がキーを押してデータや演算を入力するがコンピュータだとプログラムから読み込むので早いとか、という議論ができる。

ウ） 授業では、1個のメモリに統合することによってプログラムとデータの領域の配分が自由になることを挙げた。 そのほかにも、プログラムをデータと思って書き換えて自己改変型プログラムが作れるとか、メモリの制御回路が1つで済むとか、いろいろ考えられるだろう。

エ） 「1つ1つ」という説明は不可である。「逐次」というのは順番にという意味を含む。 順番に実行させることによって、「次の命令がどこにあるか」を指定しなくてもよい、というのが、逐次実行で得られるメリットである。

［3］ ア） 1桁の2進加算回路（フルアダー）の、入力と出力が何であって、どういう真理値表で表現されるか、という問題である。 第4回の予習ビデオ 「2進加算の原理」 を参照。

この授業では、教科書や他の本でよく出てくる「半加算器」（ハーフアダー）を2つ組み合わせて、1桁分の加算器（「全加算器」（フルアダー））を作る方法は、取り上げておらず、むしろ「全加算器の入力と出力の関係を直接に真理値表で書いてしまう」というアプローチを取っている。 もちろん、半加算器の2段構えという理解でも正しいので、その形できちんとかけた場合は正解とした。 ただ、2段構えにすると多少遅くなるだろう。

イ） 真理値表で表された論理を、AND/OR/NOT の組合せで作るという問題で、第1回「任意の論理演算とその合成」を参照。問は「どうやって導出したか説明せよ」としてあるので、その説明がないと不可。

ウ）エ） 複数桁の加算回路を、1桁の加算回路を組み合わせて作る。 第4回ビデオ「2進加減算の回路」を参照

［4］ 1つの命令の実行ステップを説明する問題。 第5回ビデオ 「命令が実行される仕組み」、「ノイマン型コンピュータ」参照

（ステップ1） メインメモリ（主記憶）上の、プログラムカウンタPC で指定されたアドレスの場所から、命令を CPU （中央処理装置）の「制御部」 （の命令レジスタ）へ、読み出す。

（ステップ2） 命令に含まれている、命令（OP）部、オペランド部などを識別し取り出す。

　回答として 「（読み出した）命令を解釈すること」 は、認めなかった。理由は、問題文をそのまま写しただけだから。

（ステップ3） 命令（OP）部・オペランド部に指定された動作をさせる信号を、演算部やメモリに出して、動作させる。

　回答として 「（解釈した）命令を実行すること」 は、認めなかった。理由は、問題文をそのまま写しただけだから。

（ステップ4） PC（プログラムカウンタ）を1つ進める。

（イ） PC（プログラムカウンタ）は、現在実行中の（または次に実行する）命令の、メモリ上のアドレスを保持するものである。

　（注意） PC 自体はアドレス（の数値）を保持するだけで、実行をするわけではない。

［5］ 第6回予習ビデオ 「アドレッシング」 を参照。 （ア） 1002 （イ） 1003 （ウ） 1005 （エ） 1005 （オ） 1001

［6］ 第6回予習ビデオ 「命令の実行性能」 参照。 ア） $1 \times 0.5 + 3 \times 0.2 + 5 \times 0.2 + 3 \times 0.05 + 5 \times 0.05 = 2.5$

イ） 「頻度が違うので掛ければよい」 という説明は不十分で、なぜ「掛ければよい」のかを説明してほしかった。 たとえば、プログラム全体でかかる時間は、それぞれの命令にかかるクロック数の総和である。 表に示された頻度で命令が出現するの

であれば、各種別の命令にかかるクロック数×その命令の実行回数になるから、これをプログラム全体の命令数で割ると、各種別の命令にかかるクロック数×その命令の出現頻度になる。 だから出現頻度による加重平均を用いる。
ウ） 1 クロックの時間＝1／(3×10^9)(秒)＝(1／3)×10^{-9}。 1 命令当たりの平均実行時間＝CPI×(1 クロックの時間)＝0．833(ナノ秒)
エ） MIPS値 ＝ (1／(1 命令当たりの実行時間))／10^6 (MIPSは10^6が単位) ＝ 1．25×10^3 (MIPS)

[7] (ア) 命令の意味の書き方は、第 7 回予習ビデオ 「COMET-II のプログラミング」 26 ページ目ぐらいを参照。
(イ) AもBも、計算結果は同じで、S＝X＋Y＋Z＋W
(ウ) 大きな違いは、プログラム実行の時に実行する命令の数が、Aは9つに対して、Bは5つで、Aの方が多い。 もしすべての命令のCPI(1 つ 1 つの命令を実行するクロック数)が同じであれば、Aの方が余分に時間がかかる(遅い)。
(注意) なお、「すべての命令のCPIが同じである」とは、問題文中には書いていない。 命令によって実行クロック数が異なるケースはあり得るので、回答するときに「もし同じであると仮定すれば」という断りを入れるのがよいだろう。 今後大学で勉強する中で、問いが完全でない問題が大いに出てくると思うので、その時はきちんと自分で設定すること。

[8] RISCとCISC 第 6 回の予習ビデオ 「CISCとRISC」 を参照のこと。
(ア) ①「命令体系の作り方についての」 ② (個々の)命令の動作の複雑さ(簡単さ) という性質。
RISC と CISC の比較を短く言うと、
RISC は、簡単な命令・簡単な体系を使って、複雑なことは命令の組み合わせて実現する
CISC は、複雑な命令・複雑な体系を使って、複雑なことを1つの命令で実現する(複雑なことをできる命令を多数用意する)
といえるだろう。 (予習ビデオ 5 ページ)
(イ) 比較の表は、予習ビデオ 27 ページ参照
(ウ) たとえば、
同じことをするのに、CISC だと1つ1つの命令のすることが多いので、全体として少ない命令ステップ数で実現できる。 RISC だと 1 つ 1 つの命令が単純な今年ができないので、全体として命令ステップ数は多くなる。 しかし、CISC は 1 つの命令にかかる時間が長くなる(複雑だから)のに対して、RISC は1つの命令の実行が速い(簡単だから)。 トータルの時間(＝命令ステップ数×1 つの命令にかかる時間)は、どちらがよいともいえない。

[9] 第 7 回の予習ビデオ 「条件分岐とIF文」 の例題を参照。
細かい点、特に表記上のコンマのあるなしなどは採点上は無視した(実際にはコンパイル時にエラーになるが)。
注意点として、レジスタもメモリも、上書きすれば値は変わるが、逆に言うと、上書きしない間は値を保持し続けている。 だから、たとえば最初に GR3 に値0を書込んでいれば、それを書き直さない限りはずっと値0を持つとしてよい。 つまり、Xと0との比較(CPA 命令)の時の0と、Xに0を代入するための0は、(もしその処理の間にGR3を書き換えていなければ)わざわざ値0を書込みなおさなくても、そのまま使ってよい。