Digital Systems & Linear Algebra

Marius Gavrilescu

1. (a) A disjunctive form is one of the form $A \vee B \vee C \vee \ldots \vee Z$ where $A, B, \ldots Z$ are of the form $a \wedge b \wedge c \wedge \ldots \wedge z$.

   This can be turned into a circuit of depth 2 by putting several many-input AND gates on the first level (to obtain $A, B, \ldots, Z$) and one many-input OR gate combining all these. The AND and OR gates can be replaced with NAND gates.

   (b) It is a sequence of chained flip-flops. The ith flip-flop feeds into the (i+1)th, while the input feeds into the first flip-flop. If we are using k flip-flops, we can obtain the last k inputs and evaluate the function using them (they are the outputs of all flip flops).

   (c) Use a SIPO shift register with 2 flip-flops as the sequential part. The combinational part would be $z = a \wedge b$ where a is the output of the first flip-flop, and b is the output of the second flip-flop.

   (d) We will use a SIPO with 3 flip-flops, outputs being a, b, c. Then o will be a fourth flip-flop defined as $o = (a \wedge b \wedge c) \vee (o \wedge (a \vee b \vee c))$ [minimal disjunctive form? just expand this to DNF?]

   (e) Same SIPO. Now $o = (a \wedge b \wedge c) \vee (o \wedge a \wedge b) \vee (o \wedge b \wedge c) \vee (o \wedge c \wedge a)$.

2. (a) Set rd to 0 (if register 0 is hardwired to have a value of 0) or to an unused register (if some register is never used otherwise) for such instructions. Or if the data memory yields 0 when MemRd is false, then this can be used in combination with register 0.

   (b) rd from 25:21, rs from 20:16, rt from $9:5$. Then R-type instructions will use rs, rt, rd and no immediate (nothing in 4:0) while I-type instructions will use rs, rd and an immediate in 9:0.

   (c) add, or are 6 bytes of opcode, 5 of rd, 5 of rs, 6 of ALU function, 5 of rt, 4 unused.
   addi, ori are 6 bytes of opcode, 5 of rd, 5 of rs, 6 of ALU function, 10 of immediate constant.
   load is 6 bytes of opcode, 5 of rd, 5 of rs, 16 unused.
   store is 6 bytes of opcode, 5 of rd (unused), 5 of rs, 6 of ALU function (unused), 5 of rt, 4 unused.

instruction	**ToAlu**	**MemWr**	**MemRd**	**ToReg**
add, or	T	F	F	F
addi, ori	F	F	F	F
load	X	F	T	T
store	T	T	F	X

   AluFunction (bits 15:10) will be "Add" for add, addi; "Or" for or,ori and unused (can be anything) for load and store.

   (d) addi has a smaller immediate (only 10 bits), whereas load and store don't have an immediate offset at all – they take the memory address (just) from a register.

(e) With a sll instruction (which has the same form as addi, ori), we can do:

```
add $at, $0, $0
ori $at, $at, n[31:30]
sll $at, $at, 10
ori $at, $at, n[29:20]
sll $at, $at, 10
ori $at, $at, n[19:10]
sll $at, $at, 10
ori $at, $at, n[9:0]
```

3. (a) Overhead for a request is relatively fixed – small blocks would be inefficient as the overhead would dwarf the actual reading time.

   (b) A benefit is the file system does not need to be rewritten for the new devices, which means fewer bugs and an easier to maintain codebase. A cost is that better filesystems could be developed that would be more efficient on such devices (such as keeping data in something else than fixed-size blocks to reduce space overhead).

   (c) The filesystem can hold files as large as the number of blocks times the size of the block (that is, the whole filesystem excluding the table of headers and the block overhead).

   (d) Keeping the number of the preceding block allows for seeking backwards within the file, whereas keepign the number of the file identifies which file a block belongs to; this information could be used for example in the implementation of a defragmenting program that would move all blocks of a file together for more efficient access.

   (e) The i-node contains a number of direct blocks (10), followed by a singly indirect block (containing pointers to direct blocks), a doubly indirect block (containing pointers to singly indirect blocks) and a triply indirect block (containing pointers to doubly indirect blocks). If a block can hold N pointers, a file can be at most $10 + N + N^2 + N^3$ blocks long.

   (f) Space overhead in UNIX system for a maximum length file is size of i-node without block pointers + 13 pointers + 1 block + (N+1) blocks + (N^2+N+1) blocks = N^3+2N^2+3N + 13 pointers + size of rest of i-node (ignored).

   Space overhead in System B for a maximum length file is 2 pointers per file + 3 pointers per block used. With a file $10 + N + N^2 + N^3$ blocks long, the overhead will be $3N^3 + 3N^2 + 3N + 32$.

   UNIX has less overhead than System B and allows faster random access of files (it is easy to find the block with a given id in UNIX, while in System B a linear search would be needed).

   UNIX system is significantly more complicated than System B, which could be a significant cost.

4. (a) Determinant is $2c^2 + 8c^2 + 7c^2 - 8c^2 - 14c - c^3 = -c^3 + 9c^2 - 14c = -c(c^2 - 9c + 14) = -c(c-2)(c-7)$. For c 0, 2 or 7 the matrix is singular.

   (b) It is the product of its diagonal.

   (c) It does not affect the determinant.

   (d) $$\begin{bmatrix} 1 & t & t^2 & t^3 \\ 0 & 1-t^2 & t-t^3 & t^2-t^4 \\ 0 & t-t^3 & 1-t^4 & t-t^5 \\ 0 & t^2-t^4 & t-t^5 & 1-t^6 \end{bmatrix}$$
   $$\begin{bmatrix} 1 & t & t^2 & t^3 \\ 0 & 1-t^2 & t-t^3 & t^2-t^4 \\ 0 & 0 & 1-t^2 & t-t^3 \\ 0 & 0 & t-t^3 & 1-t^4 \end{bmatrix}$$
   $$\begin{bmatrix} 1 & t & t^2 & t^3 \\ 0 & 1-t^2 & t-t^3 & t^2-t^4 \\ 0 & 0 & 1-t^2 & t-t^3 \\ 0 & 0 & 0 & 1-t^2 \end{bmatrix}$$

   (e) Determinant is $(1-t^2)^3$, so the matrix is not invertible for $t = 1$ and for $t = -1$.

5. (a) The rank of a matrix is the dimension of its column space. The row space is the vector space formed by the matrix's rows. The column space is the vector space formed by the matrix's columns. The null space is the set of solutions x to $Ax = 0$ if the matrix is A.

   (b) $Ax = 0 \iff \begin{bmatrix} r_1 x \\ r_2 x \\ \dots \\ r_m x \end{bmatrix} = 0$, so every vector in the null space is perpendicular to all the row vectors (thus is perpendicular to any element of the row space). We get that the row space and null space are orthogonal.

   (c) $$\begin{bmatrix} 1 & 1 & 0 & 2 \\ 0 & 1 & -1 & 1 \\ 0 & -2 & 2 & -2 \\ 0 & 1 & -1 & 1 \end{bmatrix}$$
   $$\begin{bmatrix} 1 & 1 & 0 & 2 \\ 0 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$
   A basis for the row space of A is $[1,1,0,2], [0,1,-1,1]$. The rank is 2.

   (d) The first two columns are pivots, so a basis for the column space is $[1,2,2,0]^T, [1,3,0,1]^T$.

   (e) $Ax = 0$.
   A is $\begin{bmatrix} 1 & 1 & 0 & 2 \\ 0 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$.
   $x = [-a-b, a-b, a, b]$ so a basis is $[-1,1,1,0], [-1,-1,0,1]$.

6. (a) 0.25, 0, 0.5.

   (b) A pair (λ, v) such that $Av = \lambda v$.

   (c) $\begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.25 & 0.5 & 0 \\ 0.25 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 8 \end{bmatrix} = \begin{bmatrix} 5 \\ 0.75 \\ 4.25 \end{bmatrix}$

   $\begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.25 & 0.5 & 0 \\ 0.25 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 5 \\ 0.75 \\ 4.25 \end{bmatrix} = \begin{bmatrix} 5 \\ 1.625 \\ 3.375 \end{bmatrix}$

   (d) $p(\lambda) = \begin{vmatrix} 0.5-\lambda & 0.5 & 0.5 \\ 0.25 & 0.5-\lambda & 0 \\ 0.25 & 0 & 0.5-\lambda \end{vmatrix} = (0.5-\lambda)^3 - 0.25(0.5-\lambda) = (0.5-\lambda)(\lambda^2-\lambda) = -\lambda(\lambda-0.5)(\lambda-1)$

   (e) Let $v = [5, 2.5, 2.5]^T$. Then $Av = [5, 2.5, 2.5]^T$. So 1 is an eigenvalue with corresponding eigenvector $[5, 2.5, 2.5]^T$.

   (f) Distribution vector converges to the steady state vector, $[5, 2.5, 2.5]^T$.