CPU Design,
Opcodes & Registers to Detailed Design,
Hardwired and Microcoded.

Summary of lecture notes for the lectures based on Computer Systems Organization & Architecture, chapters 6 & 7, Carpinelli, J.D, Addison Wesley; and Computer Systems, Design & Architecture, chapter ?, Heuring, V. and Jordan, H.

You should refer to the designs for Carpinelli's VS_CPU for the details of the instruction set and register set. See lecture 4, CPU design, which also covers the steps from initial outline to detailed design.

1. Control unit design.

i) Develop informal decription of the instruction set & cpu.

ii) Write the abstract RTL (formal description of programmers level model).

iii) Specify the high level data path architecture to satisfy step 2.

iv) Write the detailed (concrete) RTL to implement the abstract RTL using only register steps from (3).

v) Design logic circuits for the data-path and identify the control signals needed.

vi) Write out the set of control signals needed to implement each concrete RTL step to provide a complete set for all instructions.

vii

a) Hardcoded CU

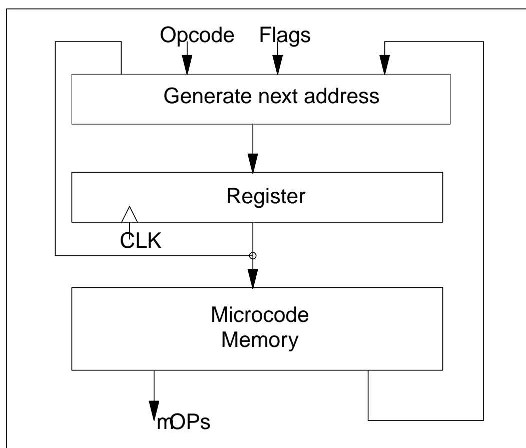
Design the sequencer and control unit, driven by a master clock, to generate the control signals. This is discussed in lecture 4.

b) Microcoded CU

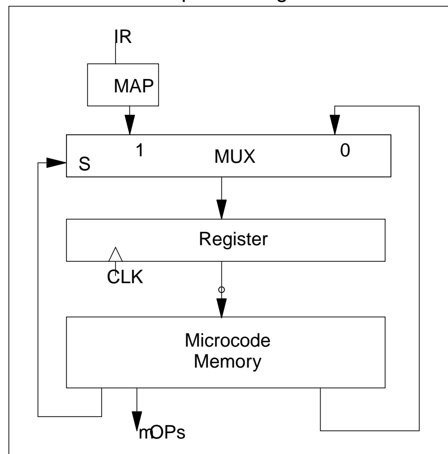
Design the microsequencer and microcode to generate the control signals .This is discussed in detail below.

2. Designing a Microsequencer

Generic microsequencer organization



Detailed microsequencer organization



Microsequencer design steps

i) Assign each state of the FSM to an address in microcode.

a) Start by assigning the address of the first state of each of the execute routines. This will decide the mapping logic. You need to be able to find a mapping from the instruction to the microcode address that ideally :

- is easy to implement for the initial state of each execute routine,
- allows for consecutive addresses for the states in fetch & execute routines, if possible, for readability and ease of debugging.

For Carpinelli's Very Simple cpu we can use the same mapping as was used for the hardwired controller, '1[IR1 IR0]0'.

b) This gives us the set of addresses shown in the first of the tables below.

State addresses for the VS_CPU

State	Address
FETCH1	0000 (0)
FETCH2	0001 (1)
FETCH3	0010 (2)
ADD1	1000 (8)
ADD2	1001 (9)
AND1	1010 (10)
AND2	1011 (11)
JMP1	1100 (12)
INC1	1110 (14)

ii) Next we have to design the microcode so that it will sequence through the states correctly and generate the correct micro-operations (μOps) at each step. Each microcode entry in the sequence will have three fields and ultimately be of the format

SEL	μOperations	Address

a) First we decide on the value of the select (SEL) field for each state. '0' will use the address field as the next address to access in the microcode, '1' will select the mapping from the Instruction. In this case SEL only needs to be '1' when we go from the end of the FETCH states (FETCH3) to the start of the current execute sequence. This is shown in the 'Partial Microcode' table below.

partial microcode

State	Address	SEL	ADDR
FETCH1	0000 (0)	0	0001
FETCH2	0001 (1)	0	0010
FETCH3	0010 (2)	1	XXXX
ADD1	1000 (8)	0	1001
ADD2	1001 (9)	0	0000
AND1	1010 (10)	0	1011
AND2	1011 (11)	0	0000
JMP1	1100 (12)	0	0000
INC1	1110 (14)	0	0000

b) We now need to deal with the μops. These are taken from the RTL for the CPU, listing each unique micro-operation such as `AR ⟵ PC`.

c) Provide each μop with a mnemonic, eg `ARPC`, as it is easier than writing `AR ⟵ PC` all the time and the mnemonic will become the bit-field name in the microcode table.

Micro-Ops and their mnemonics

Mnemonic	Micro-Operation
ARPC	$AR \leftarrow PC$
ARDR	$AR \leftarrow DR[5..0]$
PCIN	$PC \leftarrow PC + 1$
PCDR	$PC \leftarrow DR[5..0]$
DRM	$DR \leftarrow M$
IRDR	$IR \leftarrow DR[7..6]$
PLUS	$AC \leftarrow AC + DR$
AND	$AC \leftarrow AC \wedge DR$
ACIN	$AC \leftarrow AC + 1$

d) We can now complete the micro-code for our CPU by entering the appropriate value in each bit-field for the μops. For each state, when an RTL transfer must take place, we enter a '1' in the bit-field for that micro-operation and that state. This gives us the following completed table.

Horizontal microcode

State	Address	SEL	ARPC	ARDR	PCIN	PCDR	DRM	IRDR	PLUS	AND	ACIN	ADDR
FETCH1	0000	0	1	0	0	0	0	0	0	0	0	0001
FETCH2	0001	0	0	0	1	0	1	0	0	0	0	0010
FETCH3	0010	1	0	1	0	0	0	1	0	0	0	XXXX
ADD1	1000	0	0	0	0	0	1	0	0	0	0	1001
ADD2	1001	0	0	0	0	0	0	0	1	0	0	0000
AND1	1010	0	0	0	0	0	1	0	0	0	0	1011
AND2	1011	0	0	0	0	0	0	0	0	1	0	0000
JMP1	1100	0	0	0	0	1	0	0	0	0	0	0000
INC1	1110	0	0	0	0	0	0	0	0	0	1	0000

Optimized horizontal microcode

It may be possible to optimise the entries in the micro-code as shown below. If, for all states, two (or more) signals have the same value then we can use one output to drive both μops. In this case `ARDR` and `IRDR` can be replaced by a single entry `AIDR`.

Optimized horizontal microcode

State	Address	SEL	ARPC	AIDR	PCIN	PCDR	DRM	PLUS	AND	ACIN	ADDR
FETCH1	0000	0	1	0	0	0	0	0	0	0	0001
FETCH2	0001	0	0	0	1	0	1	0	0	0	0010
FETCH3	0010	1	0	1	0	0	0	0	0	0	XXXX
ADD1	1000	0	0	0	0	0	1	0	0	0	1001
ADD2	1001	0	0	0	0	0	0	1	0	0	0000
AND1	1010	0	0	0	0	0	1	0	0	0	1011
AND2	1011	0	0	0	0	0	0	0	1	0	0000
JMP1	1100	0	0	0	0	1	0	0	0	0	0000
INC1	1110	0	0	0	0	0	0	0	0	1	0000

Control signal generation

The last stage is to develop the set of actual control signals that are need. These are identical to those required in the hardwired controller as we need to implement the same set of RTLs on the same register set. Howver the logic required to generate them is considerably less than for the hardwired unit. In this instance two of the control signals are generated directly from the μops and the rest require reduced logic. The complete list is given below.

Signal	Value
ARLOAD	ARPC $\vee$ AIDR
PCLOAD	PCDR
PCINC	PCIN
DRLOAD	DRM
ACLOAD	PLUS $\vee$ AND
ACINC	ACIN
IRLOAD	AIDR
ALUSEL	AND
MEMBUS	DRM
PCBUS	ARPC
DRBUS	AIDR $\vee$ PCDR $\vee$ PLUS $\vee$ AND
READ	DRM

Vertical Microcode

One of the disadvantages of horizontal micro-code is that the table is mostly filled with zeros. Vertical microcode provides a means of encoding the micro-code in a way that compresses it. This is achieved by separating the micro-operations into fields. They are grouped into fields in a way that no more than one micro-op in a field is active during any state. A unique field value is then given to each micro-op in the field. for example, a filed with 8 micro-ops could be encoded with 3 bits. These encoded values would then be passed to a decoder. The decoder output is the same as the original horizontal microcode.

There are four basic steps to developing the vertical microcode which will demonstrated using the Very Simple CPU as an example. The bulk of the design remains the same as for the hardwired version as the same state machine has to be implemented.

1. If 2 micro-ops occur during the same state, assign them to different fields. A filed can only output one micro-operation during a cycle. If two micro-ops are to occur simultaneously, they cannot be in the same field.
2. Have a NOP in each field to cover the cases when no micro-op is to be output. A micro-op must always be output, so the NOP can be used when no micro-op is active.
3. Distribute the remaining micro-ops to make the best use of the field bits, ie, the number of micro-ops in a field should be a power of 2 if possible. Fields do not have to be the same size.
4. Group together operations that modify the same registers in the same fields. This is because 2 micro-ops cannot modify the same register simultaneously. But bear in mind the previous rule.

The VS_CPU will need at least 2 fields, M1 & M2. This decision is made after inspection of the horizontal microcode, there are 2 simultaneous micro-ops in FETCH2. Therefore we have

M1	*M2*
NOP	NOP
DRM	PCIN

Continuing to assign micro-ops to a field and ensuring that micro-ops that affect the same register are in the same field gives

M1	*M2*
NOP	NOP
DRM	PCIN
ACIN	PCDR
PLUS	ARPC
AND	AIDR

THe next step is to juggle the micro-ops around between the fields to minimize the bits required. Currently 3 bits are needed for each field totalling 6 bits. AIDR can be moved, we now require 3 bits and 2 bits for each field, saving 1 bit for every word of micro-code. In this example we can take this further by moving ARPC and PCDR as well. This way M1 now contains 8 micro-ops and M2 contains 2 requiring a total of 4 bits.

M1	
Value	Micro-op
000	NOP
001	DRM
010	ARPC
011	AIDR
100	PCDR
101	PLUS
110	AND
111	ACIN

M2	
Value	Micro-op
0	NOP
1	PCIN

State	Address	Sel	M1	M2	ADDR
FETCH1	0000	0	010	0	0001
FETCH2	0001	0	001	1	0010
FETCH3	0010	1	011	0	XXXX
ADD1	1000	0	001	0	1001
ADD2	1001	0	101	0	0000
AND1	1010	0	001	0	1011
AND2	1011	0	110	0	0000
JMP1	1100	0	100	0	0000
INC1	1110	0	111	0	0000