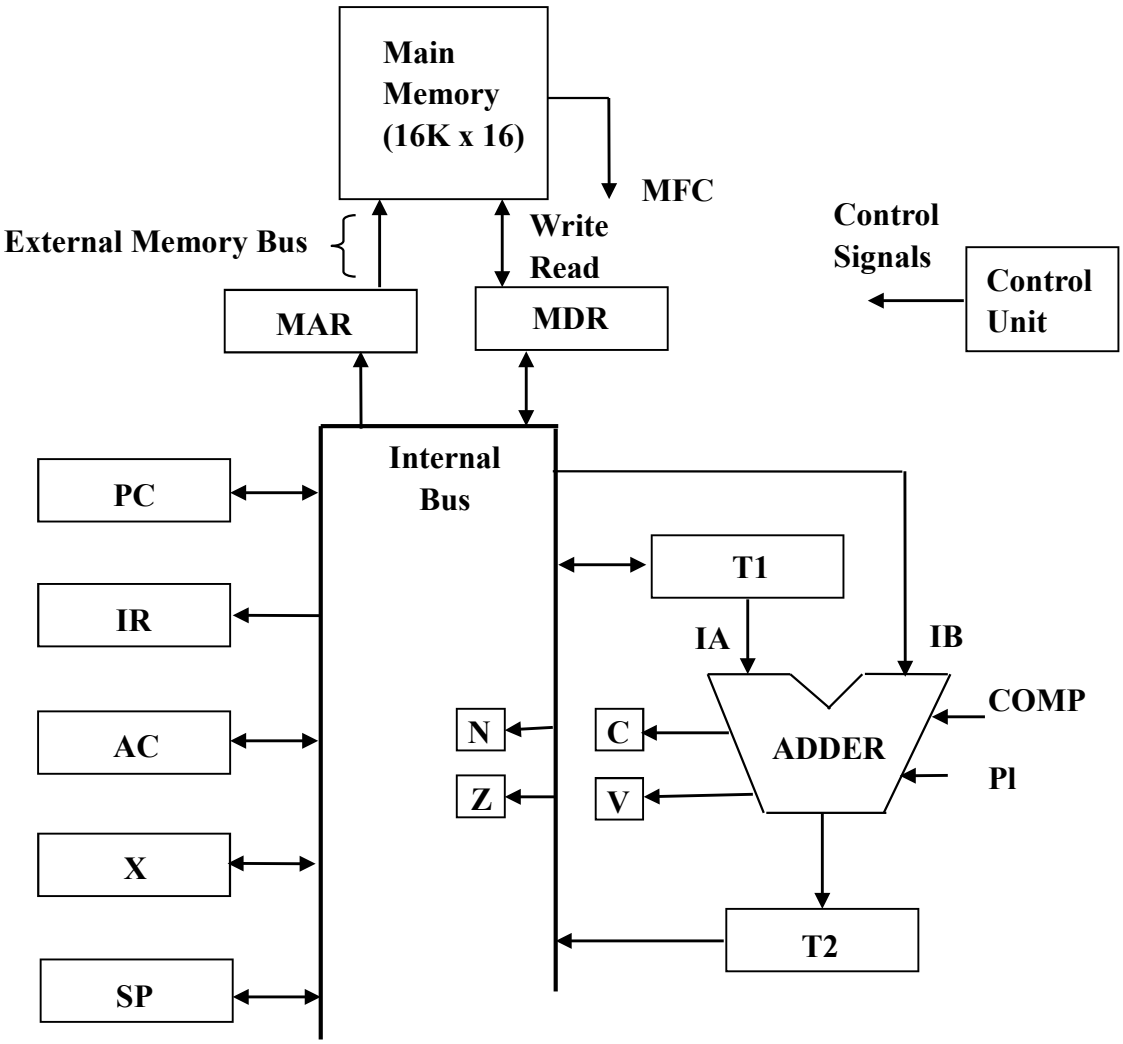
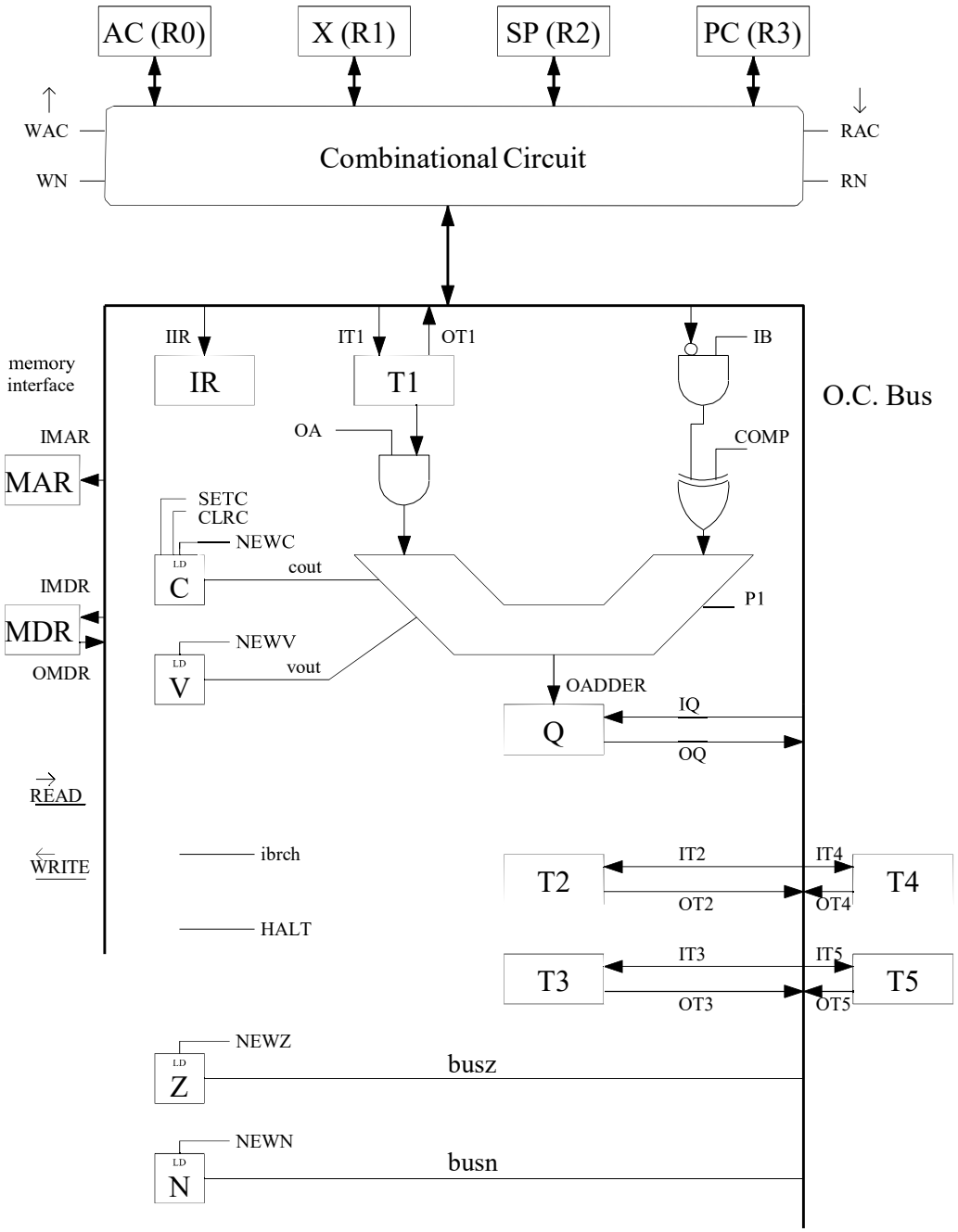
Data Path of Example Machine

ECE 5362 Handout (Sp 2017)



- IA, when asserted, causes T1 to be input to the "A" side of the adder; 0 otherwise.
- IB, when asserted, causes the BUS value to be input to the "B" side of the adder; 0 otherwise.
- COMP complements the "B" side input; Pl adds 1 to the sum.
- Memory is asynchronous; Memory Function Complete (MFC) indicates that a Read or Write has been completed.
- The word size is 16 bits.

OSIAC 3562 DATA PATHS



```
*********************************
**          AU18 ECE5362           **
**       Machine Problem 4 **
**  Youngsoo Kang, Tommy Zarick     **
*********************************

/// START OF FETCH CYCLE ///

* Fetch first word (instruction word)
st=0     rt='[pc] -> mar'  imar rac=1 rn=3
st=1     rt='[[mar]] -> mdr'        read
st=2     rt='[mdr] -> ir'  omdr iir
st=3     rt='[pc]+1 -> q'  rac=1 rn=3 ib p1 oadder
st=4     rt='[q] -> pc'             oq wac=1 wn=3
* case for special instructions or branch
cond='ir158'      value=0  nst=300
* case for single operand instructions, and DBRA (later jumps to special instructions block)
cond='ir1512'     value=0  nst=200
* case for double operand instructions
nst=100

* Fetch second word (n of src, n of dst, or offset of conditional branch)
st=5     rt='[pc] -> mar'  imar rac=1 rn=3
st=6     rt='[[mar]] -> mdr'        read
st=7     rt='[pc]+1 -> q'  rac=1 rn=3 ib p1 oadder
st=8     rt='[q] -> pc'             oq wac=1 wn=3
* case for special instructions (only conditional branch needs second word)
cond='ir158'      value=0  nst=323
* case for single operand instructions, and DBRA (later jumps to special instructions block)
cond='ir1512'     value=0  nst=201
* case for double operand instructions
nst=101

* Fetch "last" word which is a offset of DBRA instruction
st=9     rt='[pc] -> mar'  imar rac=1 rn=3
st=10    rt='[[mar]] -> mdr'        read
st=11    rt='[pc]+1 -> q'  rac=1 rn=3 ib p1 oadder
st=12    rt='[q] -> pc'             oq wac=1 wn=3
* go back to DBRA instruction
nst=341

/// END OF FETCH CYCLE ///


/// START OF HALT INSTRUCTIONS ///

*** Halt Instruction ***
st=13    halt

/// END OF HALT INSTRUCTIONS ///


/// START OF DOUBLE OPERAND INSTRUCTIONS ///

* Fetch second word if SAD is index, absolute, or immediate; or
*        if DAD is index, or absolute.
st=100
cond='ir108'      value=4  nst=5
cond='ir108'      value=5  nst=5
cond='ir108'      value=6  nst=5
cond='ir64'               value=4  nst=5
cond='ir64'               value=5  nst=5


*** Fetch src operand to T2 ***
st=101
cond='ir108'      value=0  nst=102
cond='ir108'      value=1  nst=103
cond='ir108'      value=2  nst=103
```

```
cond='ir108'      value=3  nst=108
cond='ir108'      value=4  nst=111
cond='ir108'      value=5  nst=114
cond='ir108'      value=6  nst=115
* "Trap" program for unkown addressing mode
nst=13

* Src is register mode.
st=102 rt='[ir:s] -> t2'   rac=2 it2
nst=116

* Src is register indirect or autoincrement mode.
st=103   rt='[ir:s] -> mar'rac=2 imar
st=104   rt='[[mar]] -> mdr'        read
st=105   rt='[mdr] -> t2'  omdr it2
cond='ir108'      value=2  nst=106
nst=116

* Src is Autoincrement mode.
st=106 rt='[ir:s]+1 -> q'  rac=2 ib p1 oadder
st=107   rt='[q] -> ir:s'  oq wac=2
nst=116

* Src is Autodecrement mode.
st=108 rt='[ir:s] -> t1'   rac=2 it1
st=109   rt='[t1]-1 -> q'  oa comp oadder
st=110   rt='[q] -> ir:s'  oq wac=2
nst=103

* Src is index mode (note that MDR is holding 'n' for src).
st=111   rt='[ir:s] -> t1' rac=2 it1
st=112 rt='[t1]+[mdr] -> q'oa omdr ib oadder
st=113   rt='[q] -> mar'            oq imar
nst=104

* Src is absolute mode (note that MDR is holding 'n' for src).
st=114   rt='[mdr] -> mar' omdr imar
nst=104

* Src is immediate mode (note that MDR is holding 'n' for src).
st=115   rt='[mdr] -> t2'  omdr it2
nst=116


*** Fetch dst operand to T1, and EA(dst) to MAR ***
st=116
cond='ir64'       value=0  nst=117
cond='ir64'       value=1  nst=118
cond='ir64'       value=2  nst=118
cond='ir64'       value=3  nst=123
cond='ir64'       value=4  nst=126
cond='ir64'       value=5  nst=129
* No immediate addressing for dst
* "Trap" program for unkown addressing mode
nst=13

* Dst is register mode (note that dst EA does not exist here).
st=117   rt='[ir:d] -> t1' rac=3 it1
nst=130

* Dst is register indirect or autoincrement mode.
st=118   rt='[ir:d] -> mar'rac=3 imar
st=119   rt='[[mar]] -> mdr'        read
st=120   rt='mdr -> t1'             omdr it1
cond='ir64'       value=2  nst=121
nst=130

* Dst is autoincrement mode.
st=121   rt='[ir:d]+1 -> q'rac=3 ib p1 oadder
```

```
st=122   rt='[q] -> ir:d'  oq wac=3
nst=130

* Dst is autodecrement mode.
st=123 rt='[ir:d] -> t1'   rac=3 it1
st=124   rt='[t1]-1 -> q'  oa comp oadder
st=125   rt='[q] -> ir:d'  oq wac=3
nst=118

* Dst is index mode (note that MDR is holding 'n' for dst).
st=126   rt='[ir:d] -> t1' rac=3 it1
st=127   rt='[t1]+[mdr] -> q'       oa omdr ib oadder
st=128   rt='[q] -> mar'            oq imar
nst=119

* Dst is absolute mode (note that MDR is holding 'n' for dst).
st=129   rt='[mdr] -> mar' omdr imar
nst=119

* NOTE: at this point, src operand in T2, dst operand in T1, and EA(dst) in MAR.

* Go to appropriate double operand instruction
st=130
  cond='ir1512'   value=1  nst=131
  cond='ir1512'   value=2  nst=135
  cond='ir1512'   value=3  nst=139
  cond='ir1512' value=4 nst=143
  cond='ir1512' value=5 nst=151
  cond='ir1512' value=6 nst=152
   nst=13


*** ADD instruction ***
st=131   rt='[t1]+[t2] -> q'        oa ot2 ib oadder newc newv
* If dst is register mode, skip the memory accessing microinstructions.
cond='ir64'       value=0  nst=134

* Dst is NOT register mode (to be written into memory)
st=132   rt='[q] -> mdr'            oq imdr newz newn
st=133   rt='[mdr] -> [mar]'        write
nst=0

* Dst is register mode (to be written into register)
st=134   rt='[q] -> ir:d'  oq wac=3 newz newn
nst=0


*** SUB instruction ***
st=135   rt='[t1]-[t2] -> q'        oa ot2 ib comp p1 oadder newv
* Note that above microinstruction's control line is missing newc since
*        C condition bit needs to be flipped.
cond='cout'       value=1  nst=137
st=136   setc
nst=138
st=137   clrc
* Reusing codes since storing result to dst is identical to those of ADD instruction.
st=138
cond='ir64'       value=0  nst=134
nst=132


*** MOVE instruction ***
st=139
  cond='ir64'     value=0 nst=142

* DAD is NOT register mode --> move src (residing in t2) to memory, dst EA is in MAR
st=140 rt='[t2] -> mdr'    ot2 imdr newz newn clrc clrv
st=141 rt='[mdr] -> [mar]' write
nst=0
```

```
* DAD is register mode --> move src (residing in t2) to a register
st=142   rt='[t2] -> ir:d' ot2 wac=3 newz newn clrc clrv
nst=0

*** EXG instruction ***
st=143
  cond='ir64' value=0 nst=146

* DAD is NOT register mode --> move src (residing in t2) to memory, dst EA is in MAR
st=144 rt='[t2] -> mdr' ot2 imdr
st=145 rt='[mdr] -> [mar]' write
  nst=147

* DAD is register mode --> move src (in t2) to dst register
st=146 rt='[t2] -> ir:d'   ot2 wac=3

st=147
  cond='ir108' value=0 nst=150

* SAD is NOT register mode --> move dst (residing in t1) to memory, src EA is in MAR
st=148 rt='[t1] -> mdr' ot1 imdr
st=149 rt='[mdr] -> [mar]' write
  nst=0

* SAD is register mode --> move dst (in t1) to src register
st=150 rt='[t1] -> ir:s' ot1 wac=2
  nst=0


*** OR instruction ***
st=151 rt='[t2] OR [t1] -> q' ot2 ot1 ib oadder
  nst=158

*** AND instruction ***
* logic behind this -> bubbles all around an OR gate = AND gate
* complement both input operands -> OR them together -> complement the output
st=152 rt='[t2]_comp -> q' ot2 ib comp oadder
st=153 rt='[q] -> t2' it2 oq

st=154 rt='[t1]_comp -> q' ot1 ib comp oadder
st=155 rt='[q] -> t1' it1 oq

st=156 rt='[t1] OR [t2] -> q' ot2 ot1 ib oadder
st=157 rt='[q]_comp -> q' oq ib comp oadder
  nst=158


** Store routine for OR and AND ([q] -> dst)**
st=158
  cond='ir64' value=0 nst=161

* DAD is not register mode
st=159 rt='[q] -> mdr' oq imdr clrv clrc newn newz
st=160 rt='[mdr] -> [mar]' write
  nst=0

* DAD is register mode
st=161 rt='[q] -> ir:d' oq wac=3 clrv clrc newn newz
  nst=0

/// END OF DOUBLE OPERAND INSTRUCTIONS ///



/// START OF SINGLE OPERAND INSTRUCTIONS ///

* at this point, IR has ir15_12 as 0, which includes all single instructions and DBRA.
```

```
* Fetch second word if DAD is index or absolute
st=200
cond='ir64'       value=4  nst=5
cond='ir64'       value=5  nst=5

* dst operand to T1, EA to MAR and T3:
* check addressing mode first
st=201
cond='ir64'       value=0  nst=202
cond='ir64'       value=1  nst=203
cond='ir64'       value=2  nst=203
cond='ir64'       value=3  nst=208
cond='ir64'       value=4  nst=211
cond='ir64'       value=5  nst=214
* "Trap" program for unknown addressing mode
nst=13


* register mode
st=202   rt='[ir:d] -> t1' rac=3 it1
nst=215

* register indirect mode or autoincrement
st=203 rt='[ir:d] -> mar & t3'      rac=3 imar it3
st=204 rt='[[mar]] -> mdr'          read
st=205 rt='[mdr] -> t1'             omdr it1
cond='ir64'       value=2  nst=206
nst=215

* autoincrement mode
st=206 rt='[ir:d]+1 -> q'           rac=3 ib p1 oadder
st=207   rt='[q] -> ir:d'           oq wac=3
nst=215

* autodecrement mode
st=208 rt='[ir:d] -> t1'            rac=3 it1
st=209   rt='[t1]-1 -> q'           oa comp oadder
st=210   rt='[q] -> ir:d'           oq wac=3
nst=203

* index mode (note that MDR is holding n).
st=211   rt='[ir:d] -> t1'          rac=3 it1
st=212 rt='[t1]+[mdr] -> q'oa omdr ib oadder
st=213   rt='[q] -> mar & t3'       oq imar it3
nst=204

* absolute mode (note that MDR is holding n).
st=214   rt='[mdr] -> mar & t3'     omdr imar it3
nst=204


** at this point, [dst] is in T1, EA(dst) is in MAR and T3 **

* Go to appropriate single operand instruction
st=215
cond='ir118'      value=2  nst=216
cond='ir118'      value=3  nst=217
cond='ir118'      value=4  nst=218
cond='ir118'      value=5  nst=219
cond='ir118'      value=1  nst=220
cond='ir118'      value=6  nst=221
cond='ir118'      value=7  nst=222
cond='ir118'      value=8  nst=228
* DBRA instruction jumps to the special instructions block.
cond='ir118'      value=10 nst=329
nst=13

*** INC instruction ***
st=216 rt='[t1] + 1 -> q'  oa p1 oadder newc newv
```

```
nst=229

*** DEC instruction ***
st=217 rt='[t1] - 1 -> q'  oa comp oadder newc newv
nst=229

*** NEG instruction ***
st=218 rt='0 - [T1] -> q'  ot1 ib comp p1 oadder newc newv
  nst=229

*** COMP instruction ***
st=219 rt='[t1]_comp -> q' ot1 ib comp oadder clrc clrv
  nst=229

*** CLR instruction ***
st=220 rt='0 + 0 -> q'  oadder clrc clrv
  nst=229

*** JMP instruction (not allowed for register and immediate mode) ***
st=221 rt='[t3] -> PC'  wac=1 wn=3 ot3
  nst=0

*** JSR instruction (not allowed for register and immediate mode) ***
st=222 rt='[sp] -> t1'              rac=1 rn=2 it1
st=223 rt='[t1]-1 -> q'             oa comp oadder
st=224 rt='[q] -> sp & mar'oq wac=1 wn=2 imar
* R3(PC) is already storing PC_UPD
st=225 rt='[pc] -> mdr'             rac=1 rn=3 imdr
st=226 rt='[mdr] -> [mar]'          write
st=227 rt='[t3] -> pc'                        wac=1 wn=3 ot3
  nst=0

*** TST instruction ***
st=228 rt='[t1] -> (bus)' ot1 newn newz clrv clrc
  nst=0


** store routine, assuming data we want to store is in q
**       (can be used only by CLR, INC, DEC, NEG, and COMP):
st=229
cond='ir64'       value=0  nst=232

* NOT a register mode -> put back to the [EA] that is already stored in MAR
st=230 rt='[q] -> mdr'              oq imdr newz newn
st=231 rt='[mdr] -> [mar]' write
nst=0

* if register mode -> put back to reg
st=232 rt='[q] -> ir:d'    oq wac=3 newz newn
nst=0

/// END OF SINGLE OPERAND INSTRUCTIONS ///




/// SPECIAL INSTRUCTIONS and BRANCH ///

* at this point, IR has ir15_8 as 0, which includes RTS, SC/CC, HALT, and cond branch.
st=300
*        SC/CC instruction
  cond='ir75'  value=1 nst=302
*        RTS or conditional branch, decided further at the next state (st=301).
  cond='ir76'  value=2 nst=301
*        HALT instruction
  cond='ir75'  value=0 nst=13

* Decide between RTS and conditional branch
st=301
```

```
*        if any bit of ir5_0 is 1, it's conditional branch.
  cond='ir5'  value=1 nst=322
  cond='ir4'  value=1 nst=322
  cond='ir3'  value=1 nst=322
  cond='ir2'  value=1 nst=322
  cond='ir1'  value=1 nst=322
  cond='ir0'  value=1 nst=322
*        if all bits of ir5_0 are 0, it's RTS
   nst=317


*** SS/CC instruction ***
st=302
  cond='ir4' value=1 nst=310

* clear mode
st=303
  cond='ir3' value=1 nst=304
  cond='ir2' value=1 nst=305
  cond='ir1' value=1 nst=306
  cond='ir0' value=1 nst=308
   nst=0
  st=304 rt='clear c' clrc
   nst=0
  st=305 rt='clear v' clrv
   nst=0
  st=306 rt='1 -> q' p1 oadder
  st=307 rt='[q] -> bus' oq newz
   nst=0
  st=308 rt='0 -> q' oadder
  st=309 rt='[q] -> bus' oq newn
   nst=0

* set mode
st=310
  cond='ir3' value=1 nst=311
  cond='ir2' value=1 nst=312
  cond='ir1' value=1 nst=313
  cond='ir0' value=1 nst=315
   nst=0
st=311 rt='set c' setc
  nst=0
st=312 rt='set v' setv
  nst=0
st=313 rt='0 -> q' oadder
st=314 rt='[q] -> bus' oq newz
  nst=0
st=315 rt='0 - 1 -> q' comp oadder
st=316 rt='[q] -> bus' oq newn
  nst=0


*** RTS instruction ***
st=317 rt='[sp] -> mar'    rac=1 rn=2 imar
st=318 rt='[[mar]] -> mdr' read
st=319 rt='[mdr] -> pc'    omdr wac=1 wn=3
st=320 rt='[sp]+1 -> q'    rac=1 rn=2 ib p1 oadder
st=321 rt='[q] -> sp'              oq wac=1 wn=2
  nst=0

*** Conditional Branch instructions ***
* Fetch the second word
st=322
  nst=5

* need to test (ir5 XOR ibrch). If it equals 1, do conditional branch. Otherwise, skip the rest.
st=323
  cond='ir5' value=1 nst=325
  cond='ir5' value=0 nst=324
```

```
st=324
  cond='ibrch' value=1 nst=326
  cond='ibrch' value=0 nst=0
st=325
  cond='ibrch' value=0 nst=326
  cond='ibrch' value=1 nst=0

* if branching condition is met, do the branching.
*        Note that the offset is stored in MDR.
st=326 rt='[pc] -> t1'                        rac=1 rn=3 it1
st=327 rt='[t1] + [mdr] -> q'       oa omdr ib oadder
st=328 rt='[q] -> pc'                         oq wac=1 wn=3
  nst=0


*** DBRA instruction (comes from the single op. instr. block) ***

** at this point, [dst] is in T1, EA(dst) is in MAR **

* First, test if [dst]=0 by putting it on the bus
st=329 rt='[t1] -> (bus)'  ot1
*        if [dst]!=0, decrement it first and load new PC.
*                 note that this is same as checking if [dst]!=-1 after decrement.
  cond='busz' value=0 nst=336
*        if [dst]=0, just decrement it, skip rest, and increment PC by 1 (skip the "last" word).
  cond='busz' value=1 nst=330

** Condition failed: decrement [dst], increment PC (ignore offset word), and end DBRA.
st=330 rt='[t1]-1 -> q'    oa comp oadder
*        If DAD is a register mode...
  cond='ir64' value=0 nst=333
*        If DAD is not a register mode...
st=331 rt='[q] -> mdr'              oq imdr
st=332 rt='[mdr] -> [mar]' write
  nst=334
*        If DAD is a register mode...
st=333 rt='[q] -> [ir:d]'  oq wac=3
* Increment PC
st=334 rt='[pc]+1 -> q'    rac=1 rn=3 ib p1 oadder
st=335 rt='[q] -> pc'               oq wac=1 wn=3
  nst=0

** Condition met: decrement [dst], fetch offset, and load new PC.
st=336 rt='[t1]-1 -> q'    oa comp oadder
*        If DAD is a register mode...
  cond='ir64' value=0 nst=339
*        If DAD is not a register mode...
st=337 rt='[q] -> mdr'              oq imdr
st=338 rt='[mdr] -> [mar]' write
  nst=340
*        If DAD is a register mode...
st=339 rt='[q] -> [ir:d]'  oq wac=3
* Fetch "last" word
st=340
  nst=9
** At this point, MDR is holding the offset, and R3(PC) is already storing PC_UPD
st=341 rt='[pc] -> t1'                        rac=1 rn=3 it1
st=342 rt='[t1]+[mdr] -> q'oa omdr ib oadder
st=343 rt='[q] -> pc'                         oq wac=1 wn=3
  nst=0

/// END OF SPECIAL INSTRUCTIONS and BRANCH ///
```