1ο)

Εντολές

Οι λεξεις της γλωσσας ενος υπολογιστη ονομαζονται εντολες.
Οι κύριες εντολές του MIPS κατηγοριοποιημένες σε R,I, και J τύπο είναι οι εξής

πχ .

1) I-type: addi, subi
• addi rt, rs, Immed16 Πρόσθεση καταχωρητή-σταθερής
– Σύνταξη: addi $1, $2, 100
– Functionality
• RF[rt] = RF[rs] + SignExtend32(Immed16)
• Εάν υπάρχει υπερχείληση, δημιουργία διακοπής/εξαίρεσης
• Παρατήρηση: δεν υπάρχει εντολή subi. Η αφαίρεση σταθερής γίνεται με την πρόσθεση του (-Σταθερή) χρησιμοποιώντας την εντολή addi

2) J-type: jr (jump register)
• jr rs Διακλάδωση (χωρίς συνθήκη)
– Σύνταξη : jr $12
– Η διεύθυνση προορισμού δίνεται από τα περιεχόμενα του καταχωρητή rs.
– Functionality
if (IsValidWordAddress(RF[rs]))
PC = RF[rs]
else
SignalIllegalAddressException()

Ψευδοεντολες :

Για έναν assembler είναι πολύ εύκολο να δημιουργήσει με 1-2 εγγενείς εντολές της αρχιτεκτονικής τον κώδικα για κάποιες εντολές που είναι χρήσιμες μεν στο χρήστη, αλλά δεν υποστηρίζονται εγγενώς από την αρχιτεκτονική. π.χ

1)Ψευδοεντολή move, αντιγράφει τα ένα καταχωρητή σε ένα άλλο. Υλοποιείται από τον συμβολομεταφραστή με μία addu rd, rs, 0.

2)Ψευδοεντολή li (load immediate), φορτώνει ένα καταχωρητή με μια σταθερή μέχρι 32 bits: li $2, Immediate. Εάν η σταθερή χωράει σε 16 bits, η li μεταφράζεται από τον συμβολομεταφραστή σε ori. Εάν η σταθερή χρειάζεται πάνω από 16 bits, η li μεταφράζεται σε δύο εντολές: μια lui για τα περισσότερο σημαντικά 16 bits και μια ori για τα λιγότερο σημαντικά 16 bits.

Οδηγιες :

Οι οδηγίες στον συμβολομεταφραστή MIPS ξεκινούν με μια τελεία «.». Για παράδειγμα, μια από τις κυριότερες χρήσεις οδηγιών είναι η τοποθέτηση δεδομένων στην μνήμη.

p.x
1) .text [addr] Ακολουθούν εντολές. Εάν έχει δοθεί η προαιρετική διεύθυνση, οι εντολές αρχιζουν να τοποθετούνται στην μνήμη από την διεύθυνση αυτή.

2) .data [addr] Ακολουθούν δεδομένα. Εάν έχει δοθεί η προαιρετική διεύθυνση, τα δεδομένα αρχιζουν να τοποθετούνται στην μνήμη από την διεύθυνση αυτή

2ο)

α)Δειγματοληψια: Ελεγχω ενα περιφερειακο για να δω ποτε θα μου δωσει δεδομενα
Polling, or **polled** operation, in computer science, refers to actively sampling the status of an external device by a client program as a synchronous activity. Polling is most often used in terms of input/output (I/O), and is also referred to as **polled I/O** or **software driven I/O**.

- Polling requires the CPU to actively monitor the process
- The major advantage is that the polling can be adjusted to the needs of the device
- polling is a low level process since the perepherial device is not in need of a quick response

β)Μεταφορά διαδοχικών δεδομένων από μία περιφερειακή συσκευή στην άλλη (ή συνηθέστερα μεταξύ περιφερειακής συσκευής και μνήμης επεξεργαστή) μπορεί να γίνει γρηγορότερα και πολύ φθηνά από άποψης πόρων, παρακάμπτοντας τον επεξεργαστή ο οποίος μπορεί να συνεχίσει να εκτελεί το πρόγραμμά του. Ο επεξεργαστής χρειάζεται εν γένει μόνο για να ξεκινήσει την διεργασία (και αυτό όχι πάντα). Η βασική ιδέα του DMA είναι ότι για μεταφορά διαδοχικών δεδομένων απαιτούνται τρία στοιχεία, η διεύθυνση όπου αρχίζουν τα δεδομένα στην πηγή τους (source start address), η διεύθυνση όπου αρχίζουν να εγγάφονται τα δεδομένα στον προορισμό τους (destination start address), και ο αριθμός των δεδομένων που πρέπει να μεταφερθούν. Από άποψης υλικού, η Άμεση Πρόσβαση Μνήμης (DMA) απαιτεί έναν ελεγκτή (DMA Controller), ο οποίος λειτουργεί σαν ένας μικρός και πολύ απλός επεξεργαστής.Το μόνο πού κάνει είναι με δεδομένες δύο διευθύνσεις Α και Β και ένα αριθμό λέξεων Ν, να διαβάσει διαδοχικά Ν λέξεις ξεκινώντας από το Α, και να τις αντιγράψει σε διαδοχικές διευθύνσεις ξεκινώντας από το Β.

- Works in the background without CPU intervention
- This speed up data transfer and CPU speed
- The DMA is used for moving large files since it would take too much of CPU capacity

4o)

```
p_list:
      addi  $sp, $sp, -16     #Megalwnome ti stoiva
      sw    $t0, 0($sp)       #apo8hkeuoyme tous kataxwrites
      sw    $t1, 4($sp)
      sw    $t2, 8($sp)
      sw    $ra, 12($sp)
      mul   $t0, $a3, 4       #to = 4L logw tou int *
      add   $t1, $a0, t0      #t1 = ptr + 4L
      lw    $t2, 0($t1)       #t2 = ptr[L]
      bne   $t2, $0, elseif
      move  $v0, $a3,         #timh epistrofhs sto v0(symbolo)
      j     end

elseif:
      add   $t1, $a1, $t0     #t1 = dat + 4L
      lw    $t2, 0($t1)       #t2 = dat[L]
      lw    $t1, 0($t2)       #t1 = *dat[L]
      ble   $t1, $a2, else    #an *dat[L] < x pigaine sto else
      add   $t1, $t1, $a2     #t1 = *dat[L] + X
      sw    $t1, 0($t2)       # *(dat[i]) = t1
      addi  $v0, $a3, 1       #timi epistofis L+1
      j     end

else:
      add   $t1, $t1, $a2     # t1 += X
      sw    $t1, 0($t2)       # *(dat[L]) = t1
      add   $a3, $a3, 1       # orisma L+1
      jal   p_list            # anadromiki klisi

end:
      lw    $t0,  0($sp)      # epanafora kataxwritwn
      lw    $t1,  4($sp)
      lw    $t2,  8($sp)
      lw    $ra,  12($sp)
      addi  $sp, $sp, 16      #epanafora stoivas (apodesmeusi mnim)
      jr    $ra
```

5o)

Η τελευταια γραμμη (*p) + 2 = 72 δινει συντακτικο σφαλμα επομενως δεν θα μεταγλωττιστεί άρα δε θα εκτελεσθεί άρα δεν αλλάζει κάτι στη μνήμη